The Dissertation Committee for Evgeni Krimer
certifies that this is the approved version of the following dissertation:

Improving energy efficiency of reliable massively-parallel architectures

Committee:

Mattan Erez, Supervisor

Lizy K. John

Michael Orshansky

Andreas Gerstlauer

Luis Sentis

Improving energy efficiency of reliable massively-parallel architectures

by

Evgeni Krimer, B.Sc., M.Sc.

DISSERTATION

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2012

To my wife, Tami

כי כל תורתי שלה היא

-רבי עקיבא

Acknowledgments

Now, that I am on the finish line of my journey for a PhD, I would like to acknowledge all the people who encouraged and supported me from the day I decided to take this road.

First of all, I am grateful to my advisor, Mattan Erez, for providing me with the freedom and resources to conduct high-quality research, patiently guiding me through. I am indebted to Mattan for keeping an open mind and being receptive to my ideas and directions, allowing me to follow my curiosity freely, which in turn made my experience enjoyable and valuable.

I would like to thank Patrick Chiang from Oregon State University and his VLSI research team with whom I have collaborated a lot during my work. A significant part of this dissertation is based on results obtained using the test-chip implemented and fabricated by this team. Furthermore, the test-chip allowed to verify the architectural ideas this dissertation proposes in real silicon, which is a unique opportunity I was lucky to have. I would like to thank the members of the OSU VLSI team working on the implementation of the test-chip: Robert Pawlowski, Jacob Postman, Nariman Moezzi-Madani, and especially Joseph Crop, who was always an extremely knowledgeable and helpful contact person for all my circuit-related questions.

I also acknowledge my friends in the LPH research group for interesting discussions on various topics sometimes beyond the scope of computer architecture.

I would like to thank Lizy K. John, Michael Orshansky, Andreas Gerstlauer, and Luis Sentis for serving on my dissertation committee and providing me valuable comments on this dissertation.

During the summer periods through the whole program, I had a great experience as an intern at NVIDIA, collaborating with many extremely talented and knowledgeable people from whom I learned a lot.

I would like to thank my family: My wife, Tami, left her own career to come with me through this journey, being always supporting, understanding, and encouraging. My parents supported me from the beginning through the whole process and provided help when it was needed. My parents-in-law were also extremely supportive and came to help when it was needed. I can surely say that I could not have completed this dissertation without all of their support, encouragement, and help.

In addition, I would like to thank Klara and Phil Katselnik, who did everything to make us feel home while we were away from home. From the first day I landed in Austin and all the way through, they were there for us with good advice when needed, surrounding us with a home atmosphere during the holidays as well as day-to-day occasions.

Improving energy efficiency of reliable massively-parallel architectures

Publication No. _____

Evgeni Krimer, Ph.D.
The University of Texas at Austin, 2012

Supervisor: Mattan Erez

While transistor size continues to shrink every technology generation increasing the amount of transistors on a die, the reduction in energy consumption is less significant. Furthermore, newer technologies induce fabrication challenges resulting in uncertainties in transistor and wire properties. Therefore to ensure correctness, design margins are introduced resulting in significantly sub-optimal energy efficiency. While increasing parallelism and the use of gating methods contribute to energy consumption reduction, ultimately, more radical changes to the architecture and better integration of architectural and circuit techniques will be necessary. This dissertation explores one such approach, combining a highly-efficient massively-parallel processor architecture with a design methodology that reduces energy by trimming design margins.

Using a massively-parallel GPU-like (*graphics processing unit*) baseline architecture, we discuss the different components of process variation and

design microarchitectural approaches supporting efficient margins reduction. We evaluate our design using a cycle-based GPU simulator, describe the conditions where efficiency improvements can be obtained, and explore the benefits of decoupling across a wide range of parameters. We architect a test-chip that was fabricated and show these mechanisms to work.

We also discuss why previously developed related approaches fall short when process variation is very large, such as in low-voltage operation or as expected for future VLSI technology. We therefore develop and evaluate a new approach specifically for high-variation scenarios.

To summarize, in this work, we address the emerging challenges of modern massively parallel architectures including energy efficient, reliable operation and high process variation. We believe that the results of this work are essential for breaking through the energy wall, continuing to improve the efficiency of future generations of the massively parallel architectures.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Energy consumption and power dissipation are two of the most important considerations for future processor designs. In previous technology generations, process advancements such as transistor scaling reduced energy consumption sufficiently to enable continued computational density increases, reduced form factor, and improved battery life. Today, however, transistor density is scaling faster than improvements in energy consumption because supply voltage cannot be scaled as aggressively as in the past. Furthermore, transistor dimensions continue to shrink towards the size of only a few atoms, resulting in the properties of the transistors becoming less predictable and requiring substantial guardbands, which in turn reduce efficiency and increase power dissipation. As a result of this slow scaling of efficiency, it is necessary to optimize energy efficiency and power across the entire computing system in order to continue to scale performance. This is particularly true for massively-parallel, compute-intensive processors that are already highly energy efficient from the architecture perspective alone. In this dissertation, we describe our work on improving the energy-efficiency of a massively parallel architecture by trimming the operating guardbands. All previous research that proposed guardbands reduction was done in the context of a scalar architecture

(e.g., [16]). Unfortunately, this prior work is severely limited in accommodating the demands of a massively-parallel pipeline and growing guardbands.

We base our research on a *graphics processing unit* (GPU) because it has been proven to be a very high-performance and energy-efficient design. Using this efficient architecture allows us to investigate improvements without comparing to an artificially poor baseline. To maintain a low control overhead while providing a very high performance, a GPU supports a very large number of hardware contexts. This allows a GPU to tolerate latencies to maximize throughput and utilize a large number of functional units. To improve efficiency even further, at their core GPUs depend on a single controller (sequencer) operating a large number of parallel functional units in *single instruction multiple data* (SIMD) fashion, which amortizes control decisions (Figure 1.1). Even with this very efficient design, overall performance is either bound by the power envelope or by the battery life in the case of a mobile device. Thus, further optimizing the power and energy-efficiency of

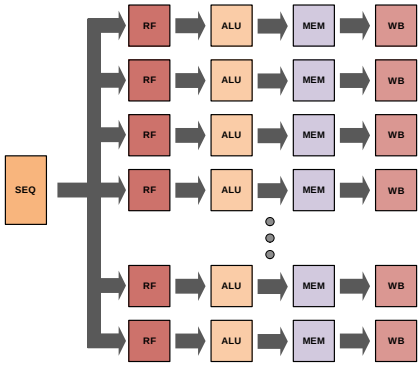


Figure 1.1: GPU-like SIMD architecture. A single controller (sequencer) operates a large number of parallel functional units.

such an architecture, as proposed in this research, is crucial to their continued performance improvement.

One significant component of the energy consumed by modern processors is in the supply voltage guardbands used to ensure reliable execution. By operating at higher than the minimum required supply voltage for a given frequency, systems are designed to tolerate the effects of varying temperature, the impact of process variation, power-supply noise, and other factors that can cause unexpected logic delays. Another source of inefficiency is that the supply voltage is set to accommodate the worst (critical) path delay within the desired cycle time, although it might be exercised very rarely. The supply voltage guardbands can be thought of either as additional time added to each cycle or as a higher than necessary voltage. Time and voltage guardbands are interchangeable because increasing the supply voltage of a CMOS circuit reduces its delay. Therefore, we do not distinguish between relaxed cycle time constraints or increased supply voltage and simply use the term guardbands.

Timing speculation [16] is a promising mechanism that improves efficiency by operating with guardbands that are smaller than those required for reliable circuit operation, in essence speculating that a small guardband is sufficient for correct operation the vast majority of time. An error detector is used to identify cases when the timing of a circuit was violated and the architecture is modified to correct the erroneous result. While this timing speculation approach has been shown to work well for simple sequential pipeline, it cannot

be directly applied to efficient parallel pipelines and to scenarios where delay variations resulting from the fabrication process are very large.

The research hypothesis for this dissertation is: **Efficient parallel architectures based on SIMD execution can be further improved by reducing operating margins, especially when considering scenarios with high process variation.**

1.1 Contributions

The main focus of this thesis is on improving the energy efficiency of massively parallel (GPU-like) architectures. Energy reduction is achieved through timing speculation, which we design specifically for this architectural style. By decreasing the supply voltage power is reduced but as a result some errors may be introduced. We analyze the tradeoffs between energy reduction and performance degradation. While power decreases with reduced voltage performance degrades as additional computations result in timing violations. We then discuss the impact of process variation, which becomes more significant with lower supply voltage, as well as with smaller feature size. We survey existing methods applicable for handling process variability and propose new approaches in the context of a parallel pipeline. Finally, we address the disadvantages of current error detection mechanisms in the context of a high variability environment and propose an alternative error detection method. We show how to use the proposed error detection mechanism to improve the

energy efficiency of high reliability SIMD architecture implementing dual redundancy.

More specifically, the main contributions described in this dissertation are:

Parallel Pipelines with Timing Speculation

- We perform a detailed analysis of timing speculation in the context of a SIMD pipeline and GPUs. We use a combination of measurements on a commercial GPU system, analytical modeling, architecture-level simulation, and circuit-level simulation to show that naively applying timing speculation to an efficient SIMD core works very poorly. An error in any functional unit stalls the entire pipeline, in effect multiplying the baseline error rate by the degree of parallelism.

- We detail the implementation of a *decoupled parallel SIMD pipeline* (DPSP), which enables efficient timing speculation in the specific context of a GPU. We describe the microarchitecture and its interaction with the GPU execution model, discuss implications and alternatives, and evaluate DPSP with detailed simulations of a GPU.

Parallel Pipelines in a High Process Variability Environment

- We survey existing error detection techniques and discuss their ability to operate under extreme process variation, such as a near-threshold operation.

- We propose a new timing-violation detection approach, *diversified duplex execution* (DDE), in which pairs of functional units, which are designed to manifest errors differently, can be configured to work together to detect violations. There are two significant advantages of the proposed approach over existing techniques. First, the suggested detector can operate even under high process variation conditions. Second, the overhead is extremely low when not in use.

- We propose a new technique, *pipeline weaving*, which provides cost-effective sparing for a SIMD pipeline to mitigate the impact of high process variation.

1.2 Dissertation Organization

The remainder of this dissertation is organized as follows: we present the background for our research in Chapter 2. Then in Chapter 3, we discuss in depth the phenomenon of process variability emerging with supply voltage reduction and advances in technology. We survey methods to address variability and propose new approaches that we evaluated with a test-chip. Later in Chapter 4, we evaluate the proposed microarchitectural mechanism to perform

in efficient manner timing and voltage speculations in SIMD (GPU-style) architectures. Then in Chapter 5, we propose and evaluate a novel timing-violation detection approach capable of operating in a high variability environment. Finally, in Chapter 6 we conclude and discuss future research directions. In addition, Appendix A provides the specification of the test-chip architected to evaluate the approaches proposed in Chapter 3.

Chapter 2

Background

In this chapter we build up the background foundations for our research. In Section 2.1, we present the baseline architecture used. Then, in Section 2.2, we briefly present the emerging challenge of process variation. We discuss process variation in further detail in Chapter 3. Later, in Section 2.3, we describe the costly design margins used today to tolerate process variation in modern circuits. We describe dynamic voltage and frequency scaling techniques in Section 2.4 and discuss an approach to trim design margins by timing speculations in Section 2.5. Finally, we present the ET^2 metric we use through this dissertation to evaluate energy-efficiency.

2.1 Architecture

The baseline architecture in this dissertation is a *single instruction multiple data* (SIMD, Figure 2.1a) [18] pipeline, which is an energy- and power-efficient parallel architecture. In a SIMD pipeline, a single instruction controls the operation of multiple functional units; all units perform the same operation, indicated by the instruction, but each unit operates on its own data (e.g., SOLOMON [48] and Illiac IV [3]). The use of a single instruction stream on

multiple data items enables efficient sharing of the costly control logic among multiple operations. SIMD pipelines are widely used in modern commercial products. *Graphics processing units* (GPUs) exploit SIMD[1] as an efficient architecture for massively parallel tasks such as graphics rendering as well as *general purpose computation on GPUs* (GPGPU) tasks, which are mainly *high-performance computing* (HPC).

Other organizations of parallel architectures include *very long instruction word* (VLIW, Figure 2.1b) [17] (e.g., Sun MAJC [74] and Transmeta Crusoe [26]) and *multiple instruction multiple data* (MIMD, Figure 2.1c) [18] (e.g, C.mmp [77] and Intel Core Duo [23]). In VLIW, similarly to SIMD, a single instruction operates multiple functional units; however, unlike in SIMD, in VLIW the instruction specifies the operation to be performed by each of the execution units (which can differ). Furthermore, different VLIW execution units often have different functional capabilities, making compiling for a VLIW architecture challenging, potentially resulting in low utilization of compute resources.

A MIMD organization, on the other hand, is usually a collection of fully functional *processing elements* (PEs) (although not necessary identical), thus making the programming task easier as each processing unit is capable

[1]Nvidia, however, refers to this execution as *single instruction multiple thread* (SIMT) because logically each SIMD execution unit and register file pair represents a different thread of control.

of running a dedicated task. However, since no control logic is shared between PEs, this organization is less energy-efficient compared to SIMD and VLIW.

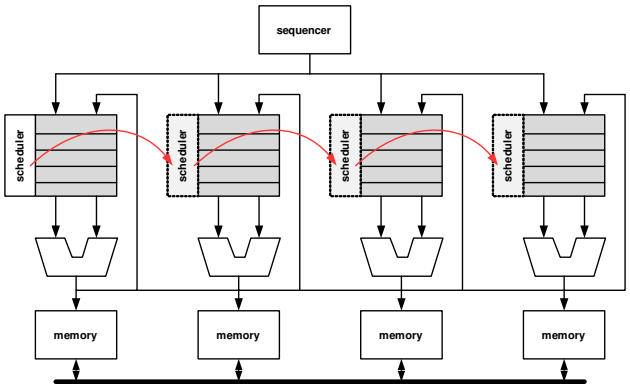
For reasons of programmability and implementation, the common approach is a hierarchy that combines different organizations in its various levels. For example, Nvidia GPUs are organized as multiple fully functional PEs (MIMD), where each PE is implementing an SIMD pipeline [55]. The same organization is expected for Intel *Knights Corner* product [68].

In this dissertation, we focus on a single PE implementing an SIMD pipeline that without loss of generality can be part of any of the mentioned organizations.

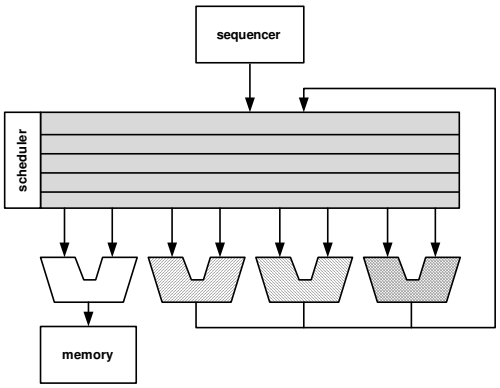
2.2 Process Variation

Fabrication of VLSI becomes extremely challenging as transistors get smaller and smaller. For example, transistor sizes are as small as the wavelength of the light source used to manufacture them, which is problematic due to fundamental optics constraints. As a result, the accuracy in doping concentrations as well as etched regions is decreased causing the properties of the manufactured transistors vary with-in a die (*intra-die variation*) and across different dies (*die-to-die variation*).

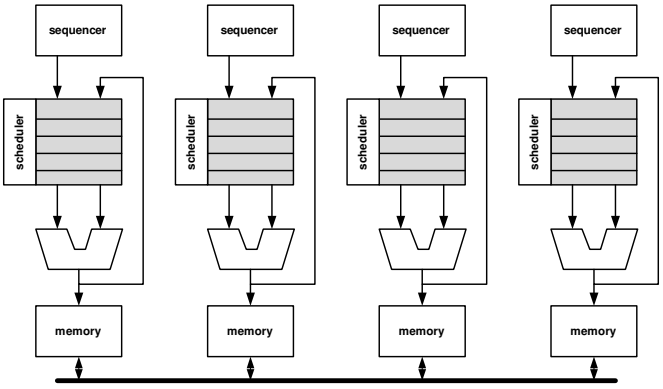
Our main interest in this dissertation is the implication of the process variability on switching delay. Decreased manufacturing accuracy results in a growing uncertainty regarding the switching time of transistors. These delay



(a) SIMD — a single sequencer operates multiple identical functional units. A shared scheduler is used, such that all units execute in lock step.



(b) VLIW — each operation explicitly controls multiple different execution units.



(c) MIMD — each execution unit is fed by a private sequencer and its operations are scheduled independently.

Figure 2.1: Different massively-parallel architectures.

fluctuations across different dies and within the same die are conventionally mitigated using guardbands (see Section 2.3).

The effects of process variation grow rapidly with decreasing feature size. Moreover, switching delay is more sensitive to the effects of process variation with decreased supply voltage, further exacerbating the impact of variation. Thus, future generations of VLSI that are projected to have both decreased feature size and reduced supply voltage are expected to have significant delay fluctuations. Even with today's technology ($\leq 65nm$) process variation is already noticeable and is a major concern when operating at low supply voltages that are in the range of V_{th} (*near-threshold operation*).

2.3 Design Margins

The traditional approach to VLSI design is conservative "worst-case" design. The supply-voltage/clock-frequency operating point is carefully chosen to accommodate the worst-case scenario of the design. Thus, todays circuits are designed to allow the completion of computation through the longest delay path (also referred to as the *critical path*) assuming worse-than-typical environmental and fabrication conditions. To do so, various design margins (also referred to as *guardbands*) are introduced. The margins can be in the form of higher supply voltage or a longer cycle time or both.

Design margins are used to tolerate temperature, noise, and clock-skew, as well as wear-out and process variation. As VLSI technology continues to advance and the feature size continues to shrink, design margins are growing with

each generation. Thus, currently, the margins consume a significant amount of the energy in VLSI circuits. Moreover, the portion of energy consumed to ensure these guardbands in future generations is projected to grow [8].

As discussed above, design margins are set to ensure worst-case computation completion within predefined time limits (cycle time). An alternative approach, however, would be to explicitly detect computation completion. A detection of computation completion allows either an *asynchronous* [27] or a *self-timed synchronous* [12] design.

The asynchronous design eliminates the need for a global clock (predefined time limits) using *hand-shake* protocols between the logic blocks (*pipe-stages*). Although several CPUs have been built using asynchronous design (e.g., AMULET [22] and MiniMIPS [59]), this approach has strong disadvantages mainly due to its complexity and incompatibility with the majority of commercial *electronic design automation* (EDA) tools [35].

Self-timed synchronous design, unlike the asynchronous design, still uses the global clock in conjunction with completion detectors. The transfer of the data between the logic blocks (*pipe-stages*) occurs on the first cycle after computation completion is detected. Synchronizing with the global clock reduces the complexity of design and validation of the self-timed synchronous design compared to asynchronous design. This approach, however, requires costly completion detectors.

Dual-rail encoding [13, 45, 65, 67] is the most widely used approach for detecting completion. Both the variable and its complement $(x, \overline{x})$ are used as the input pair and the output is a pair as well $(f, \overline{f})$; the output pair must be complementary when computation is complete. This approach introduces extremely high area and energy overheads because it requires permanently doubling the number of transistors and results in an increased wiring complexity.

An alternative completion detection approach is by *current-sensing* [14]. This method requires observing the current flowing through the supply to the on-chip logic. After the current profile to the on-chip logic has settled, the current transient will converge towards steady-state leakage current, signifying that computation has finished. Unfortunately, this type of detection has not been proven to operate correctly in the sub/near-threshold domain, where leakage current is very large relative to the switching current. In addition, this method requires challenging post-silicon calibration and is very sensitive to noise events, such as power-supply noise and other soft-error events that can cause large transient current switching.

Because of the reasons discussed above, in this dissertation we focus on the synchronous circuits and discuss alternative methods to reduce design margins.

2.4 Dynamic Voltage and Frequency Scaling

The simplest way to reduce power consumption is to employ clock throttling, known as *dynamic frequency scaling)* (DFS). DFS improves power linearly, but does not improve energy efficiency, because dynamic switching energy is unchanged and static energy per operation is actually increased. *Dynamic voltage scaling* (DVS) attempts to maintain the same throughput while increasing energy efficiency by keeping the clock frequency constant while reducing the supply voltage. As long as no timing violations occur, DVS quadratically reduces the dynamic energy consumed. *Dynamic voltage-frequency scaling* (DVFS) incorporates both techniques simultaneously, improving power cubically [9]. While DVFS can be used to improve energy efficiency it does not reduce the guardbands introduced to address input-dependent variations.

2.5 Timing/Voltage Speculation

The overhead due to design margins in VLSI circuits is constantly growing. However, elimination of the guardbands will significantly increase the likelihood of incorrect operation. In this dissertation we assume a 100% correctness requirement, as do the majority of VLSI products (CPUs, GPUs, etc.).

In order to maintain correctness while trimming the design margins (by operating at lower than the worst-case supply voltage and/or clock cycle), a new mechanism is required [16]. Such a mechanism should eliminate incorrect

outcomes. We discuss such mechanisms in further detail in Chapter 3 and propose a novel mechanism in Chapter 5.

Alternatively, some application-specific circuits, such as *digital signal processors* (DSPs), can operate correctly, with respect to their specific use in an application, even when occasional logic execution errors occur. In this case an approximate or incomplete computation can be used to improve efficiency by reducing computation correctness (or quality) [28, 32, 39, 42, 43, 50, 51, 56].

2.6 Energy-Efficiency Metric - ET^2

To evaluate energy efficiency, we use the ET^2 metric, which is the energy dissipated to perform the given computation (i.e, application run time) multiplied by the square of the time it took to execute [44]. The reason we primarily use this metric is that it isolates efficiency improvements to the architecture only and is independent of nominal voltage. A better (lower) ET^2 implies that the system will have superior efficiency for a given performance target. Note that ET^2 is the system-level equivalent for the circuit-level ED^2 (energy delay squared product). A very appealing property of ET^2 is that if one architecture has better ET^2 than another, it will exhibit higher efficiency at any chosen V_{dd}.

Other commonly-used metrics, such as energy-delay product (ED) or power-delay-squared product (PD^2) do not have this property of supply voltage invariance. Instead they directly depend on V_{dd}, which can lead to inaccurate conclusions with respect to the inherent energy-efficiency advantage of

one architecture compared to another. Because V_{dd} can be dynamically set to different values, it obfuscates architectural and operation-related contributions to energy efficiency. It is sometimes difficult to evaluate architectures using the exact same DVFS policy and impact, which muddles the analysis of architectural efficiency. This is discussed in depth by Martin [44].

Chapter 3

Process Variation

In this chapter we first discuss the sources and the impacts of process variation and classify its different sources in Section 3.1. We then present measurements of variability-induced margin requirements in Section 3.2. Then in Section 3.3, we survey the prior work on error detection and speculation mechanisms in the context of their applicability to mitigate variability-induced margins. Later, in Section 3.4 we discuss the impact of variability on SIMD architecture and propose the novel mechanisms of *decoupled parallel SIMD pipeline* (DPSP) and *pipeline weaving*. Finally, we summarize and discuss future research in Section 3.5

3.1 Process Variation Sources and Classifications

The sources of process variation are commonly classified as those that are temporal (or dynamic) and those that are spatial (or static) according to the duration and nature of their influence [2]. Static variations are due to the permanent defects produced at manufacturing that remain for the lifetime of the chip, while dynamic variations are those that can change over

Portions of this chapter were presented previously in [10] and [37].

time. The sources of dynamic variation can also be sub-classified as reversible and irreversible. For example, temperature-induced variations are dynamic-reversible, wear-related variations are mainly dynamic-irreversible, and lithography-induced variations are static.

In general, the actual sources of process variability can be described as:

- Intrinsic – atomic-level differences between devices that exist even though the devices may have an identical physical geometry and operate under identical environmental conditions. One important property that is impacted by these atomic-level differences is the *threshold voltage* (V_{th}). This effect is also referred to as *random dopant fluctuation* (RDF).
- Placement induced – different layout densities may affect the properties of both the transistors and the wires.
- Wear-induced – aging and wear-out processes change the properties of the transistors (mainly V_{th}).
- Use-induced – due to power supply variations as a result of variation in leakage, temperature, and inputs.

Both dynamic and static variation components can be either systematic or random. Random variations such as those of line edge roughness effects and voltage or random dopant fluctuations are non-predictable and can result in a wide range of sporadic variations. An example of static random variation is that of random dopant fluctuations that affect transistor threshold voltage

V_{th}. An example of dynamic random variations is voltage fluctuations, which are usually hard to predict.

Systematic variations, on the other hand, can be measured, modeled, and precisely predicted for a specific die. An example of a systematic static variation source is the chemical mechanical polishing (CMP) processes used to treat metal interconnections. Some wear-out processes can be described as an example of systematic dynamic variations, since these are changing with time but can be modeled.

3.2 Observations and Trends

Figure 3.1 shows the impact of random-dopant fluctuations (one of the process variation components) on the computation delay of a 16-bit multiplier. Monte Carlo experiments were performed using 10 random input vectors on the multiplier synthesized with a $90nm$ technology library. The top part shows the distribution of computation delays (across multiethnic random input vectors) measured with varying supply voltage using a single multiplier instance. The spread of the delays notably grows with lowering of the supply voltage. The bottom part depicts simulation results of 10 different input vectors applied to multiple instances of the multiplier simulated at the near-threshold V_{dd} $= 0.45\,V$. The properties of the transistors in each multiplier instance were randomly adjusted to emulate process variation.

We observe that in $90nm$ technology, for the circuit simulated, the distribution of delays is relatively narrow at the nominal $V_{dd} = 1\,V$. Lowering

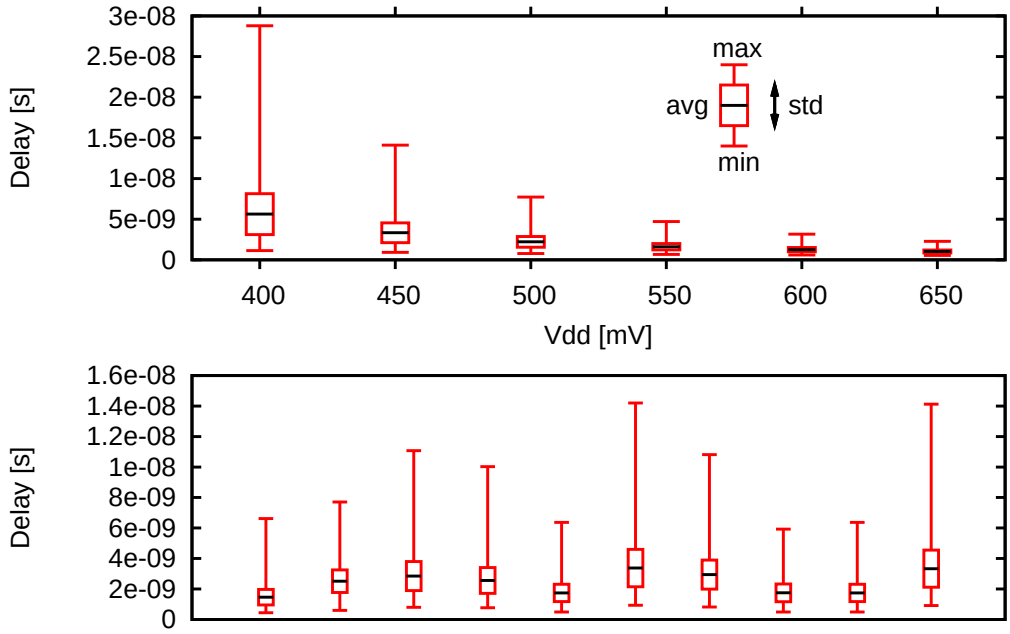


Figure 3.1: Results of random dopant fluctuation Monte Carlo experiments for the delay of a 16-bit multiplier ($90nm$, V_{th} =$0.4\,V$): worst-case delay as a function of V_{dd} (top) and delay of 10 random input vectors at $V_{dd} = 0.45\,V$ (bottom).

.

the supply voltage results in a broader delay distribution. In fact, the optimal operating point for the energy efficiency of a circuit is when the supply voltage is near the threshold voltage. This is referred to as near near-threshold operation. As shown, operating naively at near-threshold will either require large guardbands or result in low parametric yield. Furthermore, projections (Figure 3.2) predict rapid growth in delay variation for future technologies even for nominal supply voltage. Relying on design margins to tolerate process variation effects as has been done so far, will become a significant overhead and may even diminish the benefits of further technology scaling. Incorporating

speculation approaches [16] becomes a necessity to ensure further technology progress.

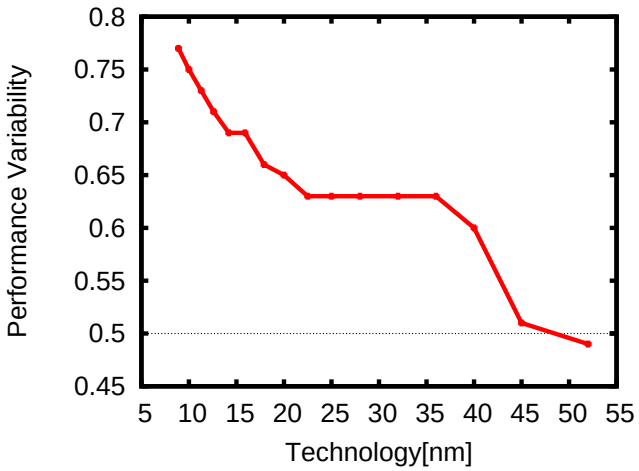


Figure 3.2: ITRS prediction [30] for circuit performance variability.

3.3 Timing Speculation for Mitigating Process Variation

Given the trend toward a growing impact of process variation on delay, speculation approaches [16] are an alternative to increasing the costly design margins. In this section we survey the mechanisms that may accommodate reducing the guardbands speculatively. Trimming the design margins may occasionally result in errors. Therefore, to maintain correctness, a mechanism that is capable of eliminating the erroneous outcomes is required. Such mechanisms usually contain detection and recovery components, where the name of each component corresponds to its purpose.

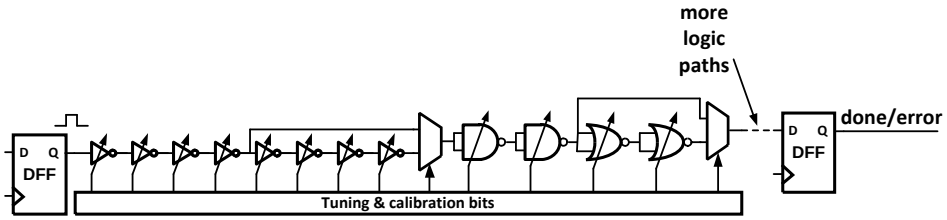


Figure 3.3: A typical tunable replica circuit.

3.3.1 Error Detection Techniques

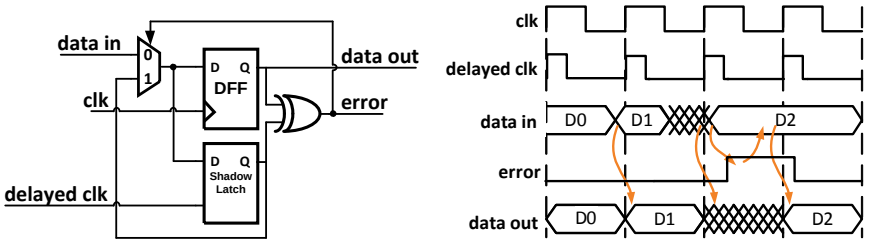
The following section surveys five error detection techniques that potentially could be used to reduce design margins in the presence of process variation. We summarize the major characteristics of these techniques later in Section 3.3.3.

3.3.1.1 Tunable Replica Circuits

One method for detecting errors due to voltage and delay speculation is the use of *tunable replica circuits* (TRCs) [76]. These circuits are composed of a number of digital cells, such as inverters, NAND, NOR gates, and wires that are tunable to a given delay time (Figure 3.3). The replicas are affected by process variations and aging in a similar way to the critical path. However, the random component of process variation will result in differences between the TRC and the actual critical path. These deferences can be resolved by tuning the TRC post fabrication. Furthermore, since the impact of environmental changes as well as wear-out/aging can vary between the TRC and the actual critical path, either recalibrations [15] and/or partial margins are required.

The major drawbacks of TRCs are the complex tuning process and the capability of detecting only the worst-case path. As such, it can accommodate the variance between different circuit instances but not the wide delay distribution corresponding to different inputs (Figure 3.1).

3.3.1.2 Razor Flip-Flops



(a) A shadow latch controlled by a delayed clock usage for error detection.

(b) Recovery from error in cycle 2.

Figure 3.4: Razor flip-flop error detection mechanism.

Razor [16] works by pairing each flip-flop within the datapath with a shadow latch that is controlled by a delayed clock. As shown in Figure 3.4, after the data propagates through the shadow latch, the output of both of the blocks is compared. If the combinational logic meets the setup time of the flip-flop, the correct data is latched in both the datapath flip-flop and the shadow latch and no error signal is set. Different values in the flip-flop and shadow latch indicate an erroneous result was propagated and then an error is detected. The possibility exists that the datapath flip-flop could become metastable if setup or hold-time violations occur. Razor uses extra circuitry to determine if the flip-flop is metastable. If so, it is treated as an error and

appropriately corrected. An important property of Razor flip-flops is that the shadow latch is designed to pick up the correct result upon the delayed clock. Therefore, a simple 1-cycle stall followed by using the correct result from the shadow latch is a valid and simple recovery option.

A proliferation of this approach, known as Razor II [11], uses a positive level-sensitive latch combined with a transition detector to perform error detection as shown in Figure 3.5. Errors are detected by monitoring transitions at the output of the latch during the high clock phase. If a data transition occurs during the high clock phase, the transition detector uses a series of inverters combined with transmission gates to generate a series of pulses that serve as the inputs to a dynamic OR gate. If the data arrives past the setup time of the latch, the detection clock discharges the output node and an error is flagged. Replacing the datapath flip-flop from Razor with a level-sensitive latch eliminates the need for metastability detection circuitry. By removing the master-slave flip-flip and metastability detector, this version shows improved power and area over Razor. However, Razor II eliminates the simple recovery option that uses a pipeline stall signal, requiring re-execution (re-computation).

When using Razor, it is important to be aware of the trade-offs that exist in achieving correct utilization of the timing window that enables Razor to properly detect errors. If a short path exists in the combinational logic and reaches the error latch before the delayed clock-edge of the computation preceding it, a false error signal could occur. To correct this, buffers are

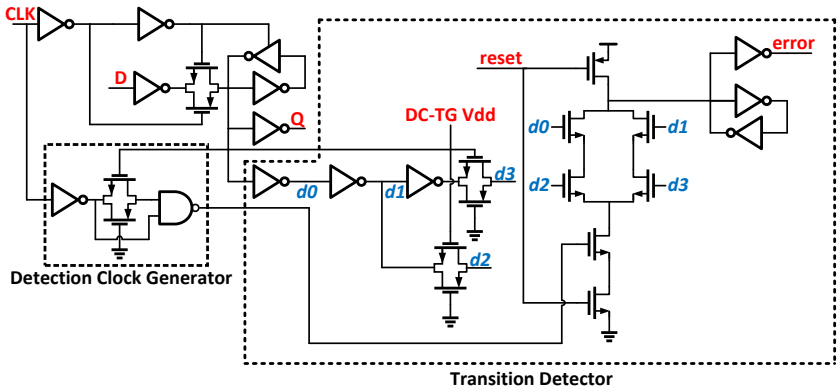


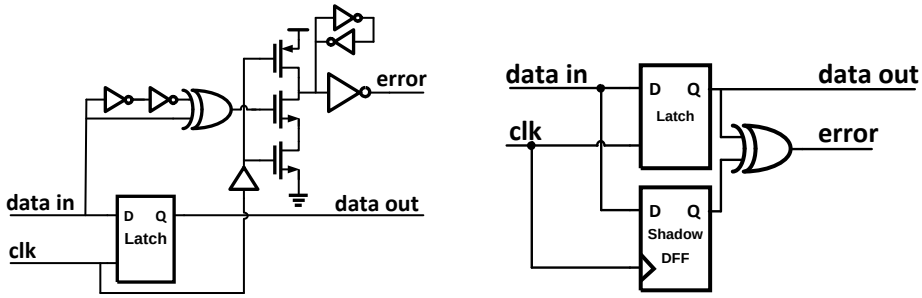
Figure 3.5: Razor II latch with detection clock generator and transition detector.

inserted in the fast paths to ensure that all paths can still be correctly caught. While this can help to guarantee that the minimum timing constraint (hold time) of the shadow latch is met, it also leads to additional area and power overheads.

3.3.1.3 Transition Detectors

A number of other methods exist that also serve as capable error detection methods. The transition detector with time-borrowing (TDTB) [4] is similar to Razor II in that it uses a dynamic gate to sense transitions at the output of a level-sensitive latch. Shown in Figure 3.6a, the TDTB differs from Razor II by using an XOR gate to generate the detection clock pulse, and the dynamic gate used in the transition detector uses fewer transistors.

Double sampling with time-borrowing (DSTB) [4] is similar to the previous circuit but with the transition detection portion replaced with a shadow



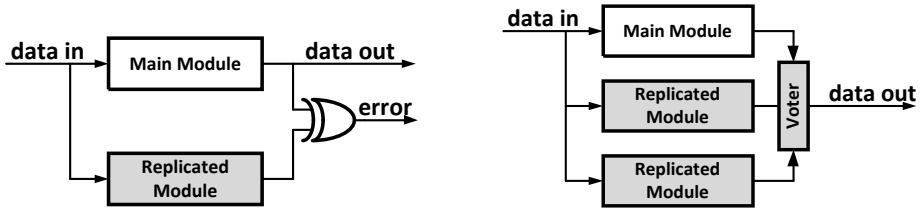
(a) Transition Detection with Time Borrowing (TDTB).

(b) Double Sampling with Time Borrowing (DSTB).

Figure 3.6: Modified Razor flip-flops - TDTB and DSTB.

flip-flop (Figure 3.6b). Like Razor, the DSTB double samples the input data and compares the datapath latch and shadow flip-flop to generate an error signal. The advantages of DSTB are that it also eliminates the metastability problem with Razor by having the flip-flop in the error path and retaining the time-borrowing feature from the transition detector. Clock energy overhead is lower than Razor since the datapath latch is sized smaller than the flip-flop used in Razor. On the other hand, unlike Razor, DTSB does not allow the option of recovery by stall.

Other approaches based on similar principles, include static and dynamic stability checkers [19]. In the static stability checker, the data is again monitored during the high clock phase using a sequence of logic gates. If the input data transitions at all during the high clock phase, an error signal is generated. The dynamic stability checker uses a series of three inverters to discharge a dynamic node in the event of a data transition during the high clock phase, generating an error signal.



(a) Dual modular redundancy (DMR). (b) Triple modular redundancy (TMR).

Figure 3.7: Modular Redundancy

3.3.1.4 Spatial Redundancy

One of the simplest forms of architectural level error-protection is *dual modular redundancy* (DMR). The protected module is replicated such that both the original module and its copy share their input signals (Figure 3.7a). Outputs are compared and a mismatch indicates an error. In case of an error, the system has to restore its last verified state and re-execute from that point. Furthermore, this approach can be extended to *triple* and n modular redundancy (TMR and nMR, respectively), utilizing majority vote methods as recovery.

While having the advantage of design simplicity in using exact replication, this approach is capable of mitigating only the random variability component. Because the systematic variability has shown a high level of correlation within a local region [20] it cannot be handled using modular redundancy.

3.3.1.5 Temporal Redundancy

Temporal, or time, redundancy is another possible method to verify computation results in the presence of speculatively trimmed design margins. This approach reduces the extra-hardware requirement compared to spatial redundancy (see Section 3.3.1.4) at the cost of extra computation time. Early work proposed re-computation with various modifications of the input data [57, 61, 66, 73]; however, these approaches have to be specifically tailored for the function unit and cannot be generalized. Identical re-execution was also proposed in the context of particle-induced soft errors [60]. Since this is a software-based approach, there is no control on which execution unit will be used to perform the re-execution, nor the exact execution order. If the operation is re-executed by the same functional unit, it may produce the same faulty result assuming the internal state of the functional unit could be modified between the executions of the two operations by another computation.

Self-imposed temporal redundancy (SITR) [47] is a hardware-based approach proposing consecutive re-execution. Thus, effectively it doubles the time allowed for the computation. To avoid significant performance drop, the result of the first operation to propagate down the pipeline is being used. The re-executed operation is used for validation only. The application of SITR to pipelined logic requires only a single comparison point after the last pipestage. That also means, however, that erroneous results may propagate all the way through the pipeline, requiring microarchitectural support.

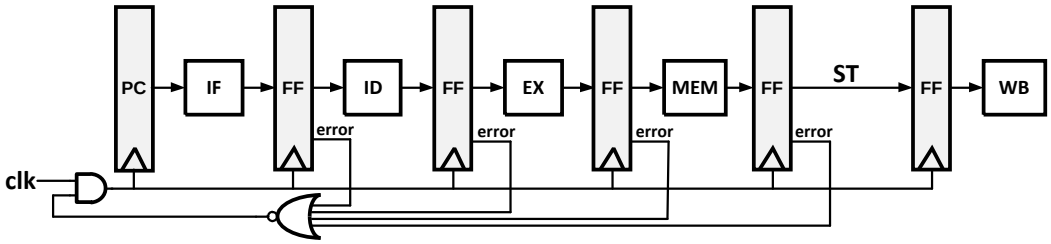
The advantage of this approach is significant in the case of underutilized functional units as in OOO (out-of-order) CPU. In the case of a highly utilized GPU, however, this approach will effectively decrease the throughput by a factor of two.

3.3.2 Error Recovery Techniques

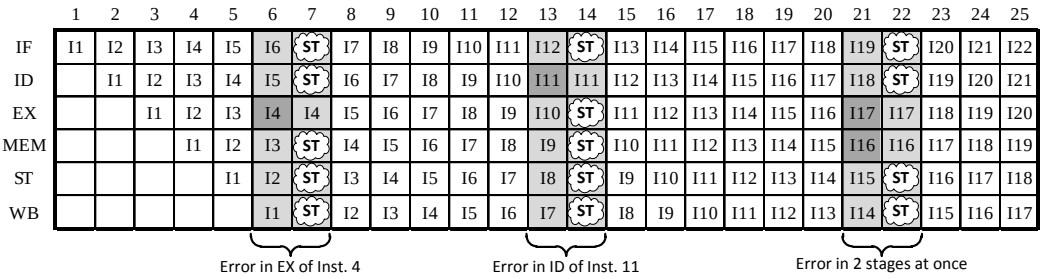
Once an error has been detected, a method needs to be in place to allow for the error to be dealt with properly. In a pipeline, later instructions may depend on the data generated by an earlier, errant instruction. Therefore, these methods need to both ensure the error is fixed (by either waiting enough time for the error to be corrected or re-executing the errant instruction while temporarily adding margins to prevent the error from repeating) and ensure the erroneous instruction does not propagate the error.

3.3.2.1 Stall

Stall by clock gating is conceptually the simplest technique to implement of all error recovery methods. Its original purpose was for saving power on unused blocks on a systems level by not clocking them when they are not used. This technique can also be adapted to error recovery by pausing all pipeline stages while waiting for the slow stage either to finish computation or to allow for the instruction to be re-executed. The pausing action ensures that other instructions do not continue to their next pipeline stage until the errant instruction is corrected. This recovery approach is most commonly paired with



(a) Pipeline modification for stall by clock gating error recovery method.



(b) Pipeline datapath with errors recovered by stalls.

Figure 3.8: A pipeline implementing a recovery by stall.

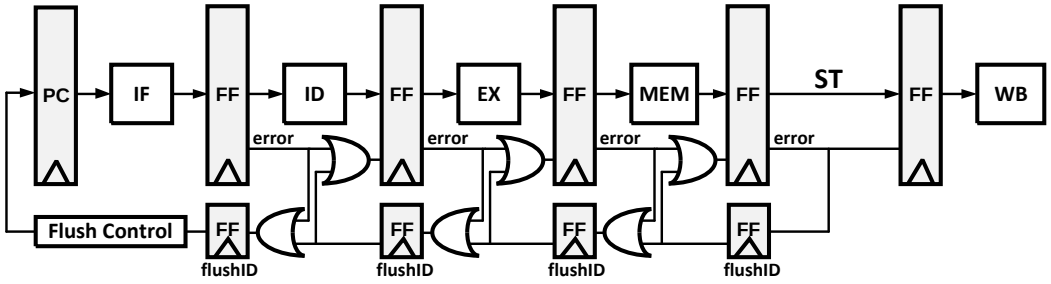
Razor flip-flops as it only works if the pipeline can be stalled before the next clock edge, before the pipeline registers are set to get new data, which can be achieved in slow systems. The stall concept is illustrated in Figure 3.8.

The primary advantage to this method is that it requires very little architectural changes as well as minimal area addition to a design compared with other methods. In order for this method to work properly, however, a stall signal needs to propagate to all pipeline stages in a very short amount of time ($\leq 50\%$ of one clock cycle when Razor circuits are used). This can be difficult to achieve across large CMOS dies where pipeline stages are several millimeters apart. We discuss the stall propagation issue in more depth in Chapter 4. Furthermore, this recovery approach can not be used in conjunction

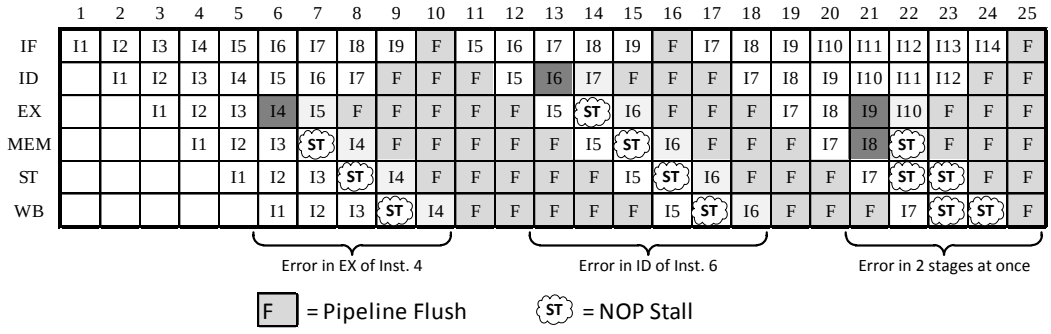
with a detection mechanism like Razor II that is unable to provide the correct outcome.

3.3.2.2 Counterflow Pipelining — Flush and Replay

Traditional *counterflow pipelining* is a microarchitecture technique that uses a bidirectional pipeline, allowing instructions to flow forward and results to flow backward. This technique made it easier to implement operand forwarding, register renaming, and, most important, pipeline flushing [16, 69]. In order to modify this technique for error recovery, a traditional pipeline is modi-



(a) Pipeline modification for flush and replay error recovery method.



(b) Pipeline datapath with errors recovered by flush and replay.

Figure 3.9: A pipeline implementing recovery by flush and replay.

fied such that only the flush signals are bi-directional. This concept is depicted in Figure 3.9. In the event of an error, flush registers begin to propagate the error signal until it reaches the flush control unit. At that point the program counter (PC) is updated with a corrected instruction pointer and the pipeline continues operation. Because the flush registers clear the pipeline in both directions as the error is propagated, there is no need to do anything other than resume execution after the error has finished propagating. An illustration of the counterflow pipeline instruction flow can be found in Figure 3.9b.

This method only requires local information to determine a stall; there is no global stall signal that needs to be computed and transmitted such as in clock gating. Unlike a stall, however, this method takes several more cycles to recover from an error as the error propagates back one stage per cycle, the pipeline needs to be flushed, and instructions need to be replayed. This can result in a serious performance degradation, especially in the context of a SIMD pipeline.

3.3.3 Summary

To summarize, Table 3.1 lists the error detection methods presented along with their limitations and applicable recovery approaches. Razor flip-flop is the only mechanism capable of recovery by stall. All other mechanisms require the costly flush and replay recovery. Furthermore, Razor flip-flop is capable of handling all sources of performance variability including both random and systematic process variation as well as input-dependent delay fluctuations.

Therefore, we assume Razor flip-flop to be the error detection mechanisms while designing and evaluating a SIMD pipeline to operate under process variation. We revisit this assumption in Chapter 5, where we discuss alternative error detection approaches.

Table 3.1: Summary of error detection mechanisms, applicable recovery approaches, and limitations.

Detection Mechanism	STALL	FLUSH	Notes
TRC		✓	Calibration issues. Cannot tolerate input-dependent variability.
RAZOR / DTSB	✓	✓	Can tolerate up to 50% performance variation. Min-delay issue.
RAZORII / TDTB		✓	
DMR		✓	Limited detection capabilities.
SITR		✓	Can tolerate up to 100% performance variation. Up to 50% throughput decrease in highly utilized GPU.

3.4 SIMD Pipeline in the Presence of Process Variation

Process-induced timing variations may be both static (requiring functional unit sparing as described later) and dynamic, input vector dependent (see Section 3.1). Prior research identified mechanisms to tolerate dynamic timing variations within a scalar pipeline [5, 16]. The idea behind these techniques is to dynamically identify timing errors and correct them in one of two ways: (1) flushing the pipeline and re-executing the instruction with more relaxed timing; or (2) stalling the pipeline for one cycle while waiting for the correct result to be generated, and then proceeding with execution. The first approach has a larger performance penalty, but is easier to implement in

high-speed and complex pipelines, while the second approach has a smaller performance and power penalty.

Unfortunately, extending either the stall or flush approach to conventional wide parallel SIMD pipelines is problematic. In a conventional parallel design, all parallel functional units operate in lock step off of a single clock, such that any error encountered in single stage will result in a stall or flush across all the functional units within that processing element. This multiplies the performance/power penalty of tolerating an error by the pipeline width. This overhead is compounded by the fact that the likelihood of an error occurring is effectively larger with a parallel pipeline because the chance of a timing violation in one functional unit is independent of the other functional units that are processing different inputs. These problems are depicted in Figures 3.10–3.12, which show preliminary calculations of the expected probability of error and expected resulting throughput as a function of the pipeline depth, pipeline width, and probability of error for a single pipeline stage. The wider the pipeline, the steeper is the degradation rate with respect to timing violation probability.

Our architectural innovations are not in this straight-forward baseline architecture, but rather in the way in which we address the challenges of extreme static and dynamic timing variations. We propose two architectural mechanisms that complement prior work on variation tolerance in parallel architectures [33]. *Decoupled parallel SIMD pipelines* extends the work on timing speculation to parallel pipelines, providing tolerance of input-dependent and

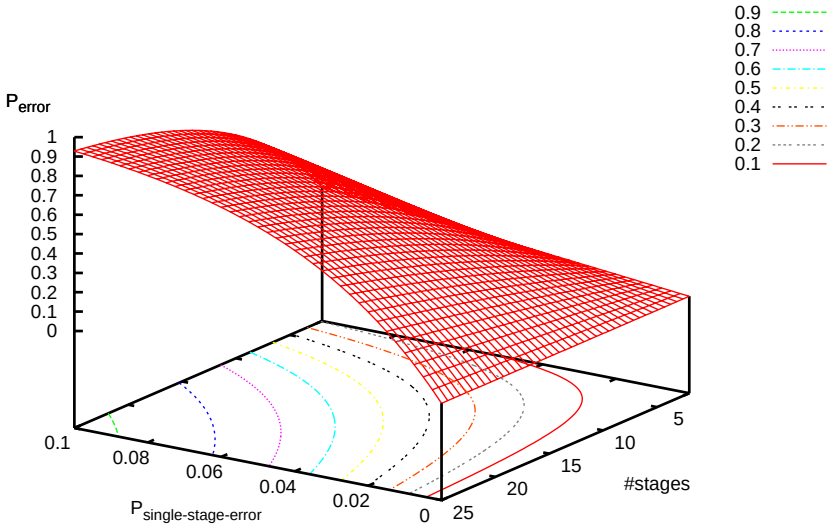


Figure 3.10: Probability of an error occurring while executing a single wide pipelined instruction (SIMD or VLIW) as a function of the probability of a single stage experiencing a timing violation and the total number of stages.

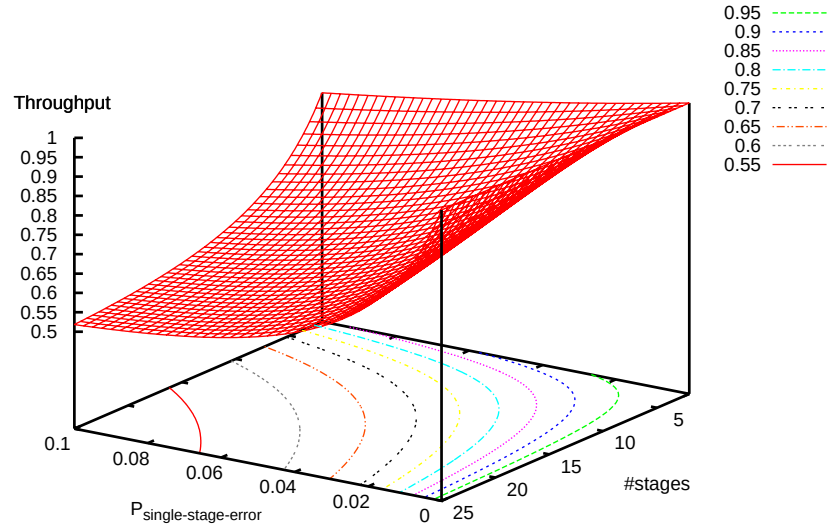


Figure 3.11: Expected throughput of a scalar pipeline as a function of its depth and the probability of a timing violation of a single stage.

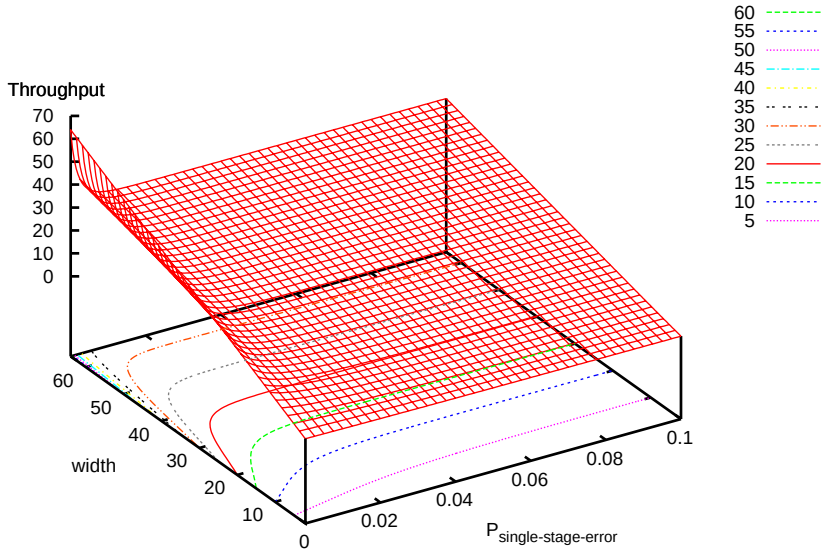


Figure 3.12: Expected throughput of a 5-stage SIMD pipeline as a function of its width (number of functional units) and the probability of a single stage experiencing a timing violation.

dynamic variations. *Pipeline weaving* provides efficient finer-grained spatial redundancy within the parallel pipeline.

3.4.1 Tolerating Dynamic Timing Variations Using DPSP

In order to address dynamic variations, where the ALU pipeline delays dynamically depend on specific instructions and data inputs, we propose a *decoupled parallel SIMD pipeline* (DPSP) microarchitecture. With DPSP, all functional units in the SIMD organization still execute the same instructions in the same order, but parallel pipelines are allowed to slip with respect to one another so that they can tolerate timing violations independently. Thus, timing violations that are input dependent will occur randomly in all parallel

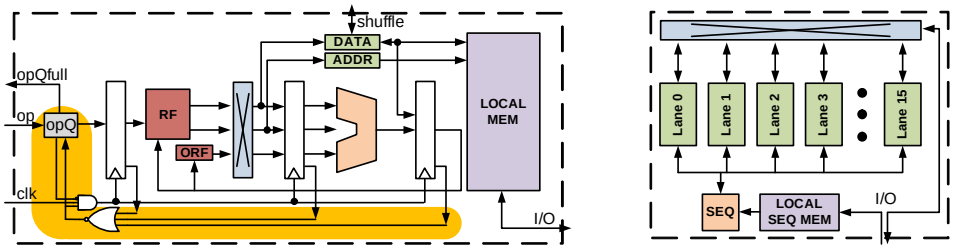
pipes, such that the average throughput will be optimal. To enable independent timing recovery in each lane, we use *decoupling queues*.

First, we replace the pipeline latch between the decode stage and the register access and execute stages of the sequential parallel pipeline with a set of shallow FIFO decoupling queues (one per SIMD lane). When a timing violation is detected in one of the lanes, only that particular lane stalls and initiates local recovery. The front-end continues to place instructions into the decoupling queues and all non-faulting lanes execute normally. If timing violations are equally distributed between operations and lanes, the average execution rate would be identical across the PE. The entire parallel pipeline stalls only if violations are not balanced between the lanes and one of the decoupling queues fills up. The DPSP concept, shown in Figure 3.13b, can be generalized by placing additional queues between additional pipeline stages to allow for re-balancing of the violations internally within each pipeline.

Results based on a simple throughput simulation indicate that DPSP can indeed maintain high parallel throughput in the face of input-dependent timing variations. We evaluate this concept of *lane decoupling* in depth in Chapter 4.

3.4.2 Pipeline Weaving for Addressing Static Timing Uncertainties

While DPSP can tolerate dynamic variations, process-dependent static variations require a different approach. For example, if a single wide-SIMD pipeline has large delay variability between ALUs, it will have poor energy



(a) Baseline PE SIMD lane. (b) Baseline PE

Figure 3.13: Overall design of a single lane and processing element. Each lane contains a fused 16-bit MADD ALU, with two inputs provided by a standard RF and the third input by a small and efficient operand register file. The highlighted paths and components within a lane enable DPSP to tolerate dynamic timing variability.

efficiency, as faster components will have higher leakage and thus need to be clocked faster to reach optimal energy-per-computation targets. Additionally, process variations can lead to non-functioning pipeline components, which may reduce both yield and performance. Prior approaches addressed this challenge at a coarse granularity, adjusting the supply voltage for entire PEs [33, 75]. Fine-grained approaches, such as voltage interpolation [41], are problematic because of the large overheads involved in providing the numerous supplies necessary for the massive number of low-power computational units. In addition to utilizing coarse-grained techniques, we propose a new complementary technique for intra-PE sparing called pipeline weaving (Figure 3.14). With this technique, we duplicate the shared fetch and decode stages of the parallel SIMD pipeline, but only add a small number of redundant parallel components. Each component is connected to two other downstream components, such that the overall parallel pipeline exhibits a weave pattern of local wires. This is an adaptation of the two-dimensional PE-array sparing design orig-

inally presented in [36] for a SIMD pipeline. Figure 3.14 also shows how this weaved pipeline design can be disabled and reconfigured around units or pipeline stages that do not meet minimum specification during configuration time. Shaded components are disabled as are all the wires that connect them (indicated with a red X in the figure). Pipeline weaving is similar to the recently presented StageNet [24] architecture, with two important differences. First, we specialize our technique for the DPSP pipeline described above and do not replicate all components of the pipeline. Second, the weaved SIMD pipeline approach does not use centralized switches as required by StageNet and instead relies entirely on local wires. This is particularly important in the sub-/near-threshold domain, because of the much higher energy cost of communication (relative to computation).

Shortly after publishing the idea of pipeline weaving [37], Gupta et al. have published a detailed analysis of a similar approach — StageWeb [25]. The StageWeb publication provides a detailed evaluation of the proposed technique and concludes that it can improve the system throughput by up to 40%. Moreover, by mitigating process variation this scheme can reduce energy consumption by 16%. Because of the StageWeb analysis publication, we decided to continue with the research in other directions.

3.5 Summary and Future Work

This chapter presents and classifies the sources of process variation. The conservative approach to process variation is to handle it by costly designs

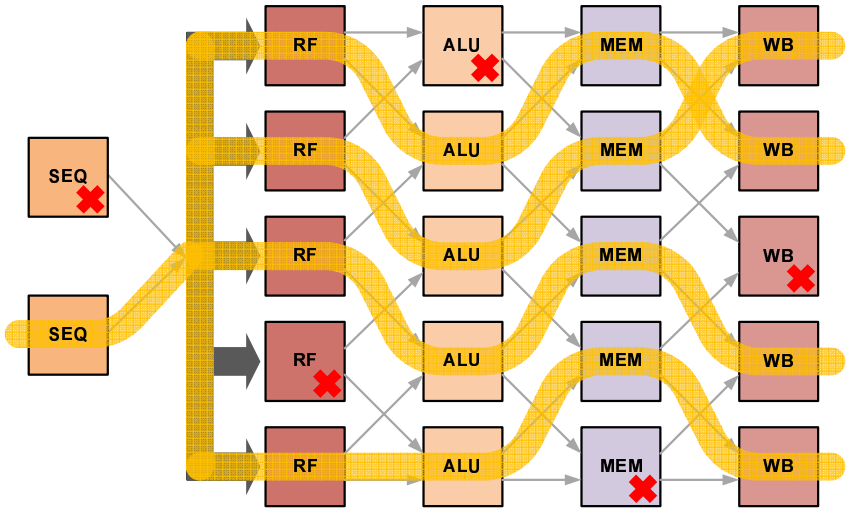


Figure 3.14: Weaved-pipeline sparing for DPSP and SIMD PEs. Shared pipeline components are replicated once and, in this example, a sparing of $\frac{1}{4}$ is added to the parallel SIMD path. Components that failed testing are marked with an X and are disabled along with their wires.

margins. An alternative is to speculatively reduce these margins, however, it may result in computation error, therefore requires dedication error detection and correction mechanisms. We survey the state of the art existing mechanisms to detect and recover from errors which are applicable to speculatively reduce design margins due to process variation and discuss their limitations.

Later, we extend this discussion to the context of the SIMD pipeline proposing novel approaches of *decoupled parallel SIMD pipeline* (DPSP) and *pipeline weaving* to mitigate the random and static components of process variation.

We leave to future research to evaluate the tradeoffs between the reduction in supply voltage and energy efficiency in the context of a SIMD pipeline

with the proposed mechanisms. Moreover, the optimum point could potentially be different based on the properties and requirements of the software.

Chapter 4

Lane Decoupling

A significant portion of the energy dissipated in modern integrated circuits is consumed by the overhead associated with timing guardbands that ensure reliable execution. Timing speculation, where the pipeline operates at an unsafe voltage with any rare errors detected and resolved by the architecture, has been demonstrated to significantly improve the energy-efficiency of scalar processor designs. Unfortunately, applying the same timing-speculative approach to wide-SIMD architectures, such as those used in highly-efficient GPUs, may not provide similar gains.

In this chapter, we make two important contributions. The first is a set of models describing a parametrized general error probability function that is based on measurements of a fabricated chip and the expected efficiency benefits of timing speculation in a SIMD context. The second contribution is a decoupled SIMD pipeline that more effectively utilizes timing speculation and recovery, when compared with a standard SIMD design that uses only conventional timing speculation. The proposed lane decoupling enables each SIMD lane to tolerate timing errors independent of other adjacent lanes, resulting

Portions of this chapter were presented previously in [38].

in higher throughput and improved scalability. We validate our models and evaluate our design using a cycle-based GPU simulator, describe the conditions where efficiency improvements can be obtained, and explore the benefits of decoupling across a wide range of parameters.

The main focus of this chapter is improving the energy efficiency of massively parallel (GPU-like) architectures through timing speculation. We explore the tradeoffs associated with timing speculation in the context of SIMD designs to develop and evaluate SIMD- and GPU-specific extensions to the timing speculation technique. We observe that naively applying timing speculation to a SIMD pipeline may perform poorly. Any error that occurs in a single functional unit stalls all the lanes of the entire SIMD pipeline, in effect multiplying the baseline error rate by the degree of parallelism and crippling the benefits of timing speculation. In response to this deficiency, we propose *decoupled parallel SIMD pipelines* (DPSP) [37]. DPSP allows limited slipping between SIMD lanes and enables each lane to tolerate errors independently, thus overcoming the deficiencies of the timing-speculative SIMD described above. DPSP also enables more efficient error recovery when a timing error is detected, as recovery is localized to a single lane. We extend the DPSP concept to a GPU and discuss GPU-specific implementation details for the first time. We also perform a detailed quantitative evaluation that is based on modeling, simulation, and fabricated circuit measurements.

To summarize the main contributions of this chapter:

- We demonstrate the potential issues of applying timing speculation to SIMD designs, arising from the fact that an error in *any* SIMD lane stalls *all* SIMD lanes. We also show that this problem may not necessarily prevent timing speculation from improving efficiency under some conditions.

- We detail the implementation of a DPSP (decoupled parallel SIMD pipeline) and recovery, in the specific context of a GPU. We describe the microarchitecture and its interaction with the GPU execution model, discuss implications and alternatives, and evaluate DPSP with detailed simulations of a GPU.

- We develop a new model for the expected probability of errors that result from timing violations when the supply voltage is reduced. This error rate model is based on a combination of analytical formulation, results from prior work [16], and measurements of a recent test-chip fabricated in a $45nm$ CMOS process [58].

- We develop a new analytical model for the potential efficiency gains of timing speculation. This includes modeling the expected execution time due to recovery overheads. Our model uses the ET^2 metric to isolate the improvements of the architecture. This is necessary because DVFS can be applied in addition to timing speculation, and using metrics

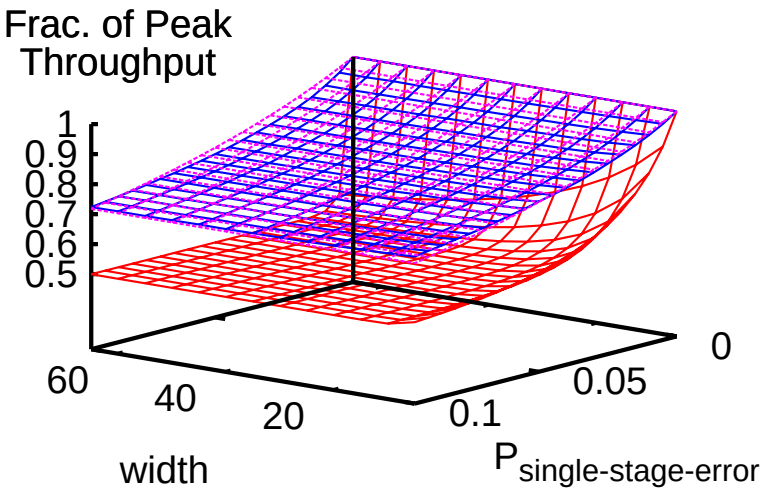
such as energy-delay product or energy-per-operation cannot distinguish between these techniques.

- We validate the efficiency model using detailed cycle-level simulations of a GPU architecture augmented with timing speculation. We then draw conclusions and present insights into when timing speculation and DPSP are beneficial, and describe expected future trends. This is the first detailed treatment of timing speculation in the context of an efficient parallel architecture built upon lock-step execution. We conclude that naive timing speculation for a GPU will only improve efficiency (ET^2) by 7.8%, whereas DPSP extends the potential gains to 10.3%.

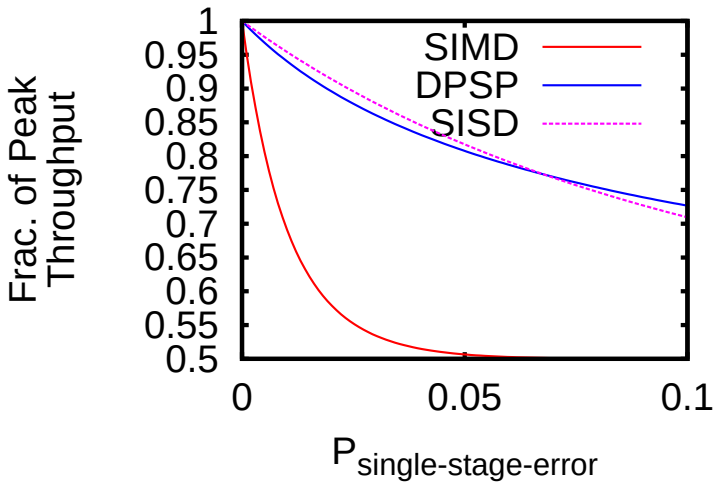
The rest of the chapter is organized as follows: we detail the implementation of timing speculation in a GPU in Section 4.1, describe our extensive methodology in Section 4.2, introduce a model for the timing-violation-induced error rate in Section 4.3, develop a model for the efficiency improvements of timing speculation and present model-based results in Section 4.4, analyze detailed microarchitecture simulation results in Section 4.5, and discuss implementation issues and summary inSection 4.6.

4.1 Proposed Architecture

Prior work, such as Razor [16], proposed to reduce the large overhead of guardbands by speculating that timing will be met in the vast majority of cases. Timing-violation detectors (based on a shadow latch) and a cor-



(a)



(b)

Figure 4.1: Expected fraction of peak throughput of a 5-stage SIMD pipeline and a 5-stage decoupled parallel SIMD pipelines with decoupling queues as a function of the total probability of error compared to a SISD pipeline (legend for both figures appears in (b)): (a) for varying SIMD width; (b) for 16-wide SIMD.

rection mechanism are introduced to enable timing-speculation and improve efficiency. To date, all analysis and experiments have been with a simple sequential pipeline. We observe that naively applying timing speculation to a SIMD core may work very poorly [37]. An error in any functional unit stalls the entire pipeline, in effect multiplying the baseline error rate by the SIMD width. Therefore even for relatively low error rates, the effective throughput decreases drastically (Figure 4.1).

To overcome the sharp decrease in throughput of the SIMD pipeline with timing speculation, we propose the *decoupled parallel SIMD pipeline* (DPSP) microarchitecture. With DPSP, all functional units in the SIMD organization still execute the same instructions in the same order, but parallel pipelines are allowed to slip with respect to one another so that they can tolerate timing violations independently. Thus, the average throughput will be optimal and relative degradation is on par with sequential (single-instruction single-data, SISD) pipelines. We describe DPSP below, apply it in the context of a GP-GPU, and evaluate the effectiveness and potential of timing speculation in this context.

To enable DPSP in a GPU, we replace the pipeline latch between the decode stage and the register access and execute stages of the sequential parallel pipeline with a set of shallow FIFO *decoupling queues* (one per SIMD lane). When a timing violation is detected in one of the lanes, only that particular lane stalls and initiates local recovery. The sequencer continues to place instructions into the decoupling queues and all non-faulting lanes exe-

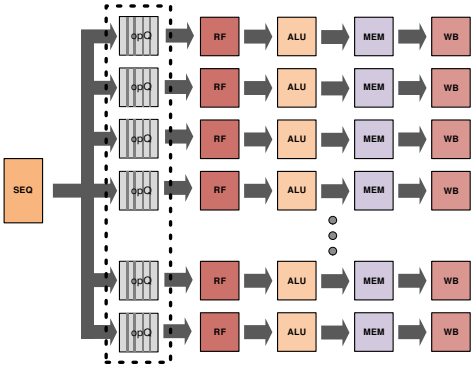


Figure 4.2: A SIMD pipeline with DPSP. DPSP components are highlighted.

cute normally. If timing violations are equally distributed between operations and lanes, the average execution rate is identical across the entire DPSP. The entire parallel pipeline stalls only if violations are not balanced between the lanes and one of the decoupling queues fills up.

The DPSP concept is shown in Figure 4.2, where the additional DPSP resources are highlighted. Note that additional decoupling queues can be placed between any two pipeline stages to allow for re-balancing of the violations internally within each pipeline. Our evaluation indicates that a single queue is sufficient to maintain high throughput in the face of input-dependent timing violation errors. Figure 4.1a compares the throughput of SIMD and DPSP for an ideal pipeline that does not stall and that experiences varying rate of random timing-violation errors (input dependent errors). Figure 4.1b shows a cross-section of the 3D curve for a 16-wide SIMD pipeline and demonstrates how DPSP experiences much lower and more gradual degradation in throughput as the likelihood of a violation grows, maintaining a $50-80\%$ better

throughput than a simple SIMD with recovery. With the evenly distributed errors used in the analysis shown in the figure, decoupling works nearly perfectly and DPSP matches the degradation curve of a sequential pipeline (SISD). We verify this property of DPSP with detailed architectural simulation and error injection in Section 4.5.

While the decoupling queues increase the effectiveness of timing speculation, the slip between DPSP lanes requires additional architectural mechanisms to ensure correct execution. In the original SIMD design, all lanes always execute in lockstep and implicitly synchronize after every instruction, but this is not true with DPSP. Any time the lanes explicitly synchronize or potentially communicate the decoupled lanes must be aligned. Current GPUs rely on SIMD pipelines for efficiency and on large degree of data parallelism to hide memory latencies. The execution model is such that communication between lanes is only guaranteed to be correct if an explicit barrier synchronization is executed [52, 71]. A barrier requires all DPSP lanes to internally synchronize to guarantee that all lanes execute the barrier. We adopt a simple solution that we show to be effective in the GPU context. When a barrier is issued, the instruction sequencer is stalled until all decoupling queues are empty. This ensures that no operations in any lane can bypass the barrier regardless of timing violations. It is possible to reduce the overhead associated with stalling the sequencer using *micro barriers* [37], however, our simulations indicate that this complexity is not necessary because barriers are executed rarely.

Even though the GPU execution model requires explicit synchronization before communication between lanes occurs, it is possible that some applications ignore this constraint. We analyzed the applications used in the evaluation and none violate the restriction. Nonetheless, we also evaluate a design in which each memory operation introduces a barrier to guarantee correctness. This barrier is handled as above and stalls the instruction sequencer until every decoupling queue is empty. As we show in our detailed evaluation (Section 4.5), synchronizing on memory accesses performs well because it maintains the original order and parallelism of memory accesses, to which some applications are sensitive.

4.1.1 Error Detection and Recovery

We utilize the conventional double-sampling [16] error detection mechanism (see Section 3.3.1.2 on page 24),utilizing a shadow latch to sample the logic output at a fixed time delay after the conventional pipeline register flip-flop. In the case an error is detected (latched outputs do not match output at delayed clock), the system recovers by stalling the pipeline for one cycle and restarts using the correct outputs of the shadow latch. Stalling for one cycle is sufficient time to allow the pipeline stage that did not meet timing to settle to a correct value. For this type of recovery to succeed, the stall signal must propagate to all pipeline stages before they overwrite the shadow latch. While this is likely not feasible in modern CPUs, and perhaps even in a wide-SIMD design, it is simple to achieve with DPSP. Because of the decoupling queues, the error

signal is required to propagate only within a single lane, which is small and can be traversed rapidly. We successfully implemented this recovery policy in our $45nm$ test-chip with DPSP, even though we were unable to accommodate single-cycle stalls with traditional SIMD. Enabling this simple single-cycle recovery mechanism is an important advantage of DPSP, because alternative recovery schemes require energy- and time-consuming pipeline flushes [16]. When comparing DPSP to SIMD, however, we assume SIMD can recover in a single cycle as well.

Note double-sampling can detect timing violations only as large as half of the cycle time. Therefore, because we set the nominal supply voltage to be V_{nom}, the supply voltage V_{dd} can be reduced down to only V_{min}, such that $\frac{T(V_{min})}{T(V_{nom})} = \frac{3}{2}$, where T is the critical path delay. V_{min} can be determined using the Alpha power-law model [63]. Figure 4.3 shows the ratio of $\frac{V_{min}}{V_{nom}}$ for various technologies based on [30]. This minimum voltage also ensures that recovery is possible by stalling for a single cycle.

4.1.2 Implementation Overheads

There are two sources of overheads that need to be addressed: the timing speculation mechanism and the DPSP structures. Because there is a range of overhead values for various timing speculation mechanisms reported in the literature [6, 11, 16**?**], we choose to present the entire range of overheads, up to a conservative 15% energy overhead, as a result of the timing speculation mechanisms. To estimate the energy consumption of the DPSP decoupling

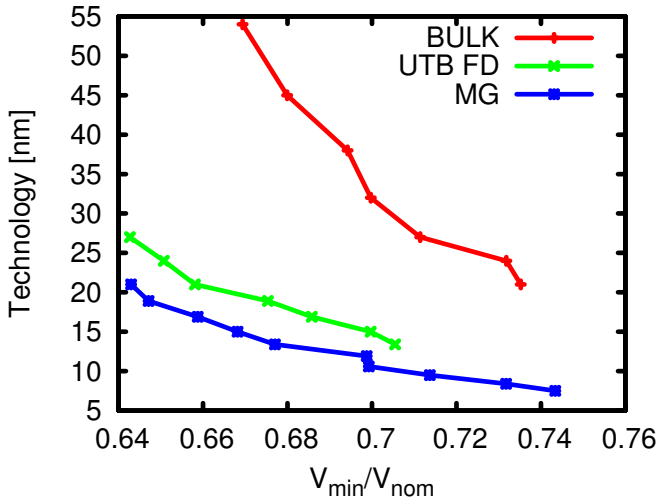


Figure 4.3: Projected $\frac{V_{min}}{V_{nom}}$ based on ITRS [30]: BULK represents planar bulk CMOS, UTB FD represents ultra-thin body fully depleted SOI CMOS, and represents multi-gate CMOS (e.g., FinFETs).

queues we use Orion 2.0 [31]. A 4 entry 32-bit wide FIFO queue with a single write and a single read port is estimated to consume $\sim 0.285pJ$ using $65nm$ node and scales down to $\sim 0.15pJ$ in $32nm$. Compared to $\sim 255pJ/op$ in a modern GPU [54], this overhead is negligible and is not visible in our results.

4.2 Methodology

The goal of our evaluation is to study the potential energy-efficiency improvements possible using timing speculation in a wide-SIMD architecture, and demonstrate the advantage of using the proposed DPSP technique. To that end, we develop a set of models for estimating timing violation error rates,

performance in the presence of such errors, and estimated energy efficiency improvements. The models are based on measurements of a manufactured chip prototype, circuit simulations, and architectural simulations. Using these models, we can quickly explore the design space and gain architectural insight. We then perform detailed architectural simulations to validate our model-based predictions.

We explain our model for error-rate as a function of V_{dd} in Section 4.3. Our methodology is to measure the distribution of delays of a functional unit, and use this distribution to determine the fraction of computations that will result in an error for a particular supply voltage. We scale the delays using the Alpha power-law model [63] and identify the cutoff based on the nominal operating frequency; we then scale the supply voltage to improve energy-efficiency for a set frequency. The initial delay distributions are based on measurements of a 16-bit multiplier fabricated in a $45nm$ IBM SOI CMOS process. We also use the error-rate results presented by Ernst et al. [16] for an 18-bit multiplier implemented in an FPGA and a 32-bit Kogge-Stone adder simulated in SPICE. We use functional units as exemplary circuits that often determine the critical path in SIMD pipelines.

The above methodology is based on a few example measurements, where each measurement is of a different type of implementation (test-chip fabrication, simulation, and FPGA-prototyping). This results in three very different error-rate functions for developing a model that captures the key

characteristics of these functions, enabling us to explore a larger design space than just these three measurements alone.

Our methodology for modeling tradeoffs in energy, explained in Section 4.4, uses this error-rate model to synthesize an analytical model that combines the impact of timing errors on performance with established models for energy consumption [9, 72]. We use this model to show results across a wide range of parameters. We then validate our model for execution time using a detailed cycle-based architectural simulation using GPGPU-sim [1] (ver. 2.1.2), which we enhanced to support DPSP and error injection Section 4.5. The corresponding energy model for the simulated GPU processor is based on the one presented in [29].

Table 4.1 summarizes the parameters of the simulated GPU architecture. This configuration is a good match for the processor used in NVIDIA's Quadro FX5800 and GTX280 GPUs. Note that the SIMD pipeline in each core has 8 lanes and that each operation is repeated 4 times for an instruction vector/SIMD length of 32.

4.3 Error Probability Model

In this section, we present our model for the probability of errors that result from both reducing the voltage guardband and speculating that the circuit will meet timing constraints. We show the error probability as V_{dd} is changed for three functional unit components and then develop a simple model

Table 4.1: Simulated architecture properties.

Number of Shader Cores	30
Threads in Warp	32
SIMD Pipeline Width*	8×4
Number of Threads / Core	1024
Number of CTAs / Core	8
Shared Memory / Core	16KB
Constant Cache / Core	8KB (2-way set assoc. 64B lines LRU)
Texture Cache / Core	8KB (2-way set assoc. 64B lines LRU)
Number of Memory Channels	8
L1 Cache	none
L2 Cache	none
Main Memory	GDDR3
Bandwidth per Memory Module	4 Bytes/Cycle
DRAM Request Queue Capacity	32
Memory Controller	Out-of-Order (FR-FCFS)
Branch Divergence Method	Immediate Post Dominator
Warp Scheduling Policy	Round Robin
Interconnect Network	Crossbar
Number of MSHRs per / Core	64

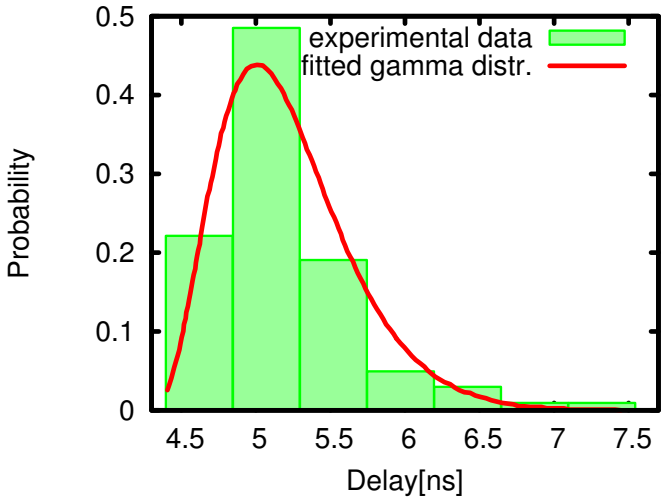


Figure 4.4: Measured delays and fitted distribution function.

that generalizes these three profiles. This model allows us to explore a much larger space of tradeoffs between the error rate and the supply voltage.

The first circuit we analyze is a 16-bit multiplier fabricated in the $45nm$ IBM SOI process. The multiplier is part of a chip manufactured to test the DPSP idea [37], but due to a combination of design and fabrication issues we are only able to measure delays with this spin of the chip. We measure the circuit delays at a single supply voltage ($V_0 = 0.53\,V$) and extrapolate the results to the full delay distribution by fitting the data to an analytical Gamma function, as suggested by Kay and Pileggi [34]. We then use the Alpha power-law model [63] to scale the analytical model of the delays to a range of supply voltages (we use $V_{th} = 480mV$ and $\alpha = 1.3$). We then calculate the error probability as a function of V_{dd} for a fixed target frequency. The target frequency was chosen to exhibit no errors at nominal V_{dd}. Next, the error probability is generated from the fraction of inputs that require a delay greater than the target cycle time (τ) for a varying supply voltage. This is shown in Equations 4.2 – 4.3.

$$P(delay = t) = \frac{(t-c)^{a-1}}{b^a \Gamma(a)} e^{-\frac{t-c}{b}} \tag{4.1}$$

Table 4.2: Gamma distribution parameters

a	4.6124
b	2.10×10^{-10}
c	4.24×10^{-9}

Table 4.3: Fit quality metrics

Kolmogorov-Smirnov	0.1579
Anderson-Darling	1.8276
Chi-Squared	22.074

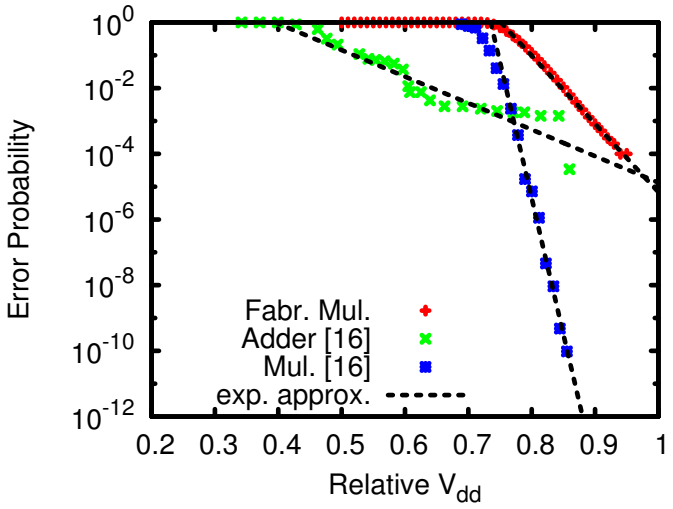


Figure 4.5: Error rate as function of supply voltage V_{dd} for various circuits.

$$F_{V_{dd}}(t) = P(t \leq \tau | V_{dd}) = F_{V_0}\left(t \cdot \frac{V_0}{(V_{dd} - V_{th})^{\alpha}} \cdot \frac{(V_0 - V_{th})^{\alpha}}{V_{dd}}\right) \tag{4.2}$$

$$P_{error}(V_{dd}) = P(t > \tau | V_{dd}) = 1 - P(t \leq \tau | V_{dd}) = 1 - F_{V_{dd}}(t) \tag{4.3}$$

Figure 4.5 depicts the error probabilities as a function of relative supply voltage for all three functional unit components . We use relative V_{dd} ($\frac{V_{dd}}{V_{nom}}$), in order to generalize our results across multiple CMOS processes, as each component was designed for a different nominal V_{nom}. The probabilities for the fabricated multiplier were based on the power-law methodology described above. For the 18-bit multiplier and 64-bit adder, we use the probabilities

reported by Ernst et al. [16]. These probabilities were derived from measurements of an FPGA implementation of the multiplier and SPICE simulations targeting $0.18\mu m$ for the adder.

Figure 4.5 also shows the exponential trendline for each of the circuits. This exponential function is a good approximation for the error probability within a range of V_{dd} values, in order to develop a simple model for the probability of a timing speculation error as a function of relative V_{dd}. Our model includes the "slope" of the probability curve (the exponent) S and the relative voltage V_{maxerr} at which the error probability reaches 1. While this model predicts a non-zero probability of error at the nominal V_{dd}, this error is negligible and we exclude it for simplicity. (Equation 4.4). Table 4.4 summarizes the model parameters for the three circuits we evaluated.

Table 4.4: Model parameters and R^2 for the circuits evaluated.

	V_{maxerr}	S	R^2
Adder [16]	0.395545254	18.58	0.9491
Multiplier [16]	0.735367316	190.7	0.9935
Measured Fabricated Multiplier	0.751256185	47.82	0.9967

$$P_{error}\left(\frac{V_{dd}}{V_{nom}}\right) = \begin{cases} 0 & \text{if } \frac{V_{dd}}{V_{nom}} = 1 \\ e^{S\cdot\left(V_{maxerr}-\frac{V_{dd}}{V_{nom}}\right)} & \text{if } V_{maxerr} \leq \frac{V_{dd}}{V_{nom}} < 1 \\ 1 & \text{if } \frac{V_{dd}}{V_{nom}} < V_{maxerr} \end{cases} \tag{4.4}$$

4.4 Model-Based Energy Efficiency Evaluation

We now address the main issue of this chapter and evaluate the relationship between the supply voltage, timing speculation errors and recovery, and the resulting energy efficiency of a traditional SIMD pipeline and a DPSP-enabled design. We first present a model for energy efficiency that abstracts many architectural details and enables a rapid exploration of the tradeoff space. We validate this model and show results from a detailed cycle-based architectural simulator in Section 4.5. We construct this energy-efficiency model by evaluating the impact of a fixed-frequency voltage scaling on execution time, dynamic power, and leakage power.

4.4.1 Execution Time

As the supply voltage is lowered, the probability of an error grows for any given computation. After each error that occurs, the pipeline must recover, thereby increasing execution time. As explained in Section 4.1, our assumed recovery mechanism inserts a single-cycle bubble into the pipeline and uses the stable and correct computation available at the end of the additional cycle. To ensure that this mechanism is correct, we do not lower the voltage below V_{min}, which is the voltage at which the longest path takes 50% longer than with nominal V_{nom} (Section 4.1.1).

Another requirement for our recovery mechanism is that the error signal must be propagated in under a cycle. This is relatively easy to do with DPSP, since the error signal is local to a single lane and only needs to propagate up

to each lane's decoupling queue. In a traditional SIMD design, It would be extremely difficult to propagate the error signal within the same cycle across all lanes. Nevertheless, we assume that it is possible, in order to compare the recovery mechanisms with our analysis and the possible advantages of DPSP.

Accurately modeling the impact of errors on execution throughput requires accounting for the effects of all architecture features, such as memory-related stalls, resource conflicts, and explicit synchronization. To develop a simple model for quickly exploring the design space and drawing insights, we defer the evaluation of these other effects to Section 4.5, and start with a simple model of executing a single (SIMD) instruction per cycle.

In a traditional lock-step SIMD pipeline with timing speculation, each error event delays execution by one cycle on every lanes. To model the impact on DPSP, we use the initial results reported in [37], such that the impact of errors on DPSP throughput can be approximated as the throughput of a scalar pipeline with the same error rate. While we show full simulation results with error injection later in the chapter, Figure 4.6 shows that approximating a DPSP as a scalar pipeline is a valid assumption. This figure compares the simplistic DPSP model, with the simulated impact of errors, on the throughput of an application that executes a single SIMD instruction/cycle. Hence, a simple compute-bound loop is not hampered by memory operations and dependencies.

Given our previous model that describes the probability of error as a function of V_{dd} ($P_{error}(V)$), we can now model the throughput and exe-

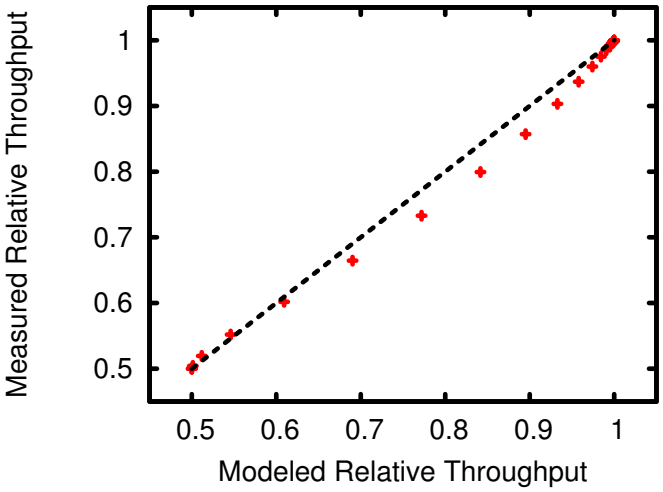


Figure 4.6: Measured relative DPSP throughput (in GPGPU-sim) vs. predicted (via model) for a synthetic compute-bound kernel.

cution time. Equation 4.5 summarizes the throughput model, where $N = simdwidth{\cdot}pipedepth$ is for a traditional SIMD, accounting for the increased error probability when any lane stalls all other lanes. For DPSP, $N = pipedepth$, which is an approximation of DPSP as a scalar pipeline. Next, we derive execution time as the inverse of throughput (Equation 4.6), assuming that the application is perfectly parallelized. Again, note that we validate these assumptions for model simplification in Section 4.5.

$$\Theta\left(P_{error}, N\right) = \left(1 - P_{error}\right)^{N} \cdot 1 + \left(1 - \left(1 - P_{error}\right)^{N}\right) \cdot \frac{1}{2} = \frac{1}{2} + \frac{1}{2}\left(1 - P_{error}\right)^{N} \tag{4.5}$$

$$T\left(V_{dd}\right) = \frac{1}{\Theta\left(V_{dd}\right)} \tag{4.6}$$

4.4.2 Energy

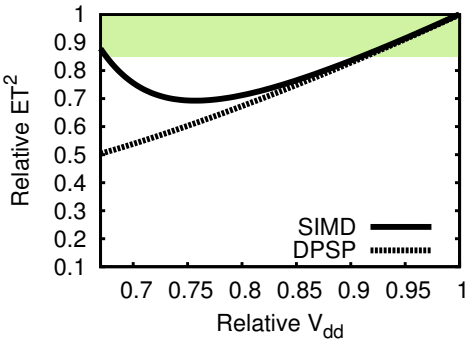
We derive the relative energy consumed as a function of V_{dd} by considering dynamic and static energy components separately. Dynamic energy simply scales with V_{dd}^2, because no additional computation takes place (the recovery mechanism stalls the pipeline for one cycle). We model the static energy scaling using the execution time model described above and approximate the change in static power as directly proportional to V_{dd}, as suggested by [72]. This linear approximation is reasonable because we vary V_{dd} over a relatively small range that is within 40% of the nominal supply voltage. Equation 4.7 represents the energy consumption at V_{nom}, where ϕ is the fraction of dynamic energy versus the total energy for the entire cycle time at V_{nom}. The relative energy as voltage is scaled is shown in Equation 4.8, where $T()$ is the delay as a function of V_{dd} (Equation 4.6). The scaled time is necessary because static energy increases as the computation time increases, due to integrated leakage energy. When comparing ET^2, we also consider that the baseline design does not require timing speculation. Because we do not precisely know the overhead for a GPU pipeline (see Section 4.1.1), we consider overheads in the range of $1 - 15\%$.

$$E\left(V_{nom}\right) = E_{nom} = \underbrace{\phi E_{nom}}_{\text{dynamic}} + \underbrace{\left(1-\phi\right) E_{nom}}_{\text{static}} \tag{4.7}$$

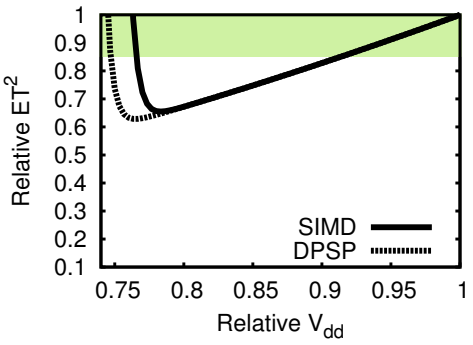
$$\frac{E\left(V_{dd}\right)}{E_{nom}} = \frac{\phi E_{nom} \frac{V_{dd}^2}{V_{nom}^2} + \left(1-\phi\right) E_{nom} \frac{V_{dd}}{V_{nom}} \frac{T(V_{dd})}{T(V_{nom})}}{E_{nom}} = \phi \frac{V_{dd}^2}{V_{nom}^2} + \left(1-\phi\right) \frac{V_{dd}}{V_{nom}} \frac{T\left(V_{dd}\right)}{T\left(V_{nom}\right)} \tag{4.8}$$

4.4.3 Model-Based Comparison

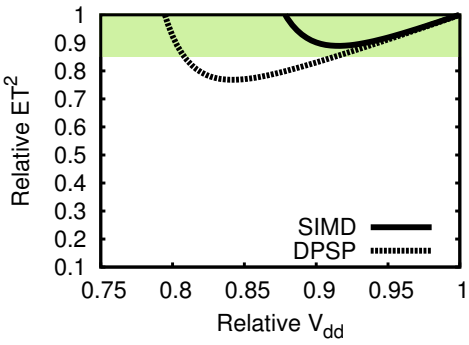
Figure 4.7 compares how ET^2 scales for non-speculative SIMD, speculative SIMD, and DPSP, as a function of V_{dd} assuming each of the three error probability functions for the three analyzed components (Section 4.3). We choose $\phi = 0.8$ (80% dynamic energy) as in the McPAT power estimation tool [40]. We normalize ET^2 to that of timing-speculation-enabled SIMD that is operated non-speculatively with DVFS. ET^2 does not change with DVFS as time increases while energy decreases and is a constant in the figure ([44]). Because DVFS can be applied on top of speculation, which is why the ET^2 metric is particularly relevant here. We normalize to a design with timing-speculation circuits to ease comparison between the implementation options, and indicate a range of non-speculative SIMD ET^2 on the figure. The bottom of the range corresponds to a very conservative overhead of 15% for implementing timing speculation (recent work reported overheads of $1-2\%$ [4, 11]. The lower the overhead the smaller the relative reduction in ET^2 for removing a design that does not include these circuits. Note that the speculation



(a) Adder [16]



(b) Multiplier [16]



(c) Measured fabricated multiplier

Figure 4.7: ET^2 as a function of V_{dd} for the three evaluated error profiles: (a) adder [16], (b) multiplier [16], and (c) measured fabricated multiplier; the estimated ET^2 for DVFS with no speculation hardware is shown by the shaded regions.

mechanism limits the lowest operating supply voltage to roughly 70% of V_{dd}, as discussed in Section 4.1.1. In some cases this lowest possible safe voltage limits the potential benefits of timing speculation.

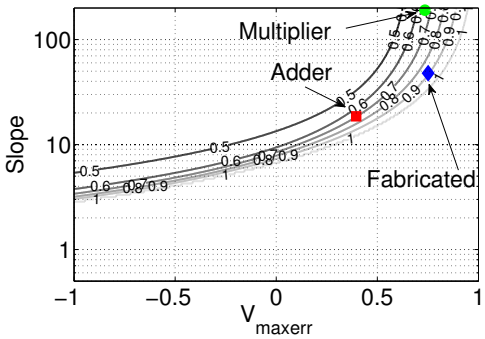
There are two important observations about these results. The first is that for each of the functional units examined, timing speculation can pro-

vide improvements in efficiency. Even without DPSP, the simulated adder and FPGA multiplier exhibit about 15% and 20% improvements respectively compared to a non-speculative baseline, even when considering very conservative speculation implementation overheads of 15%. The measured fabricated multiplier however, exhibits lower gains with no DPSP because its error function has a gentle slope and quickly reaches a probability of 1, leaving less opportunity for beneficial speculation. The large differences in potential arises from the different error function properties. The voltage at which error probability becomes 1 has the strongest influence, with a lower voltage increasing the potential gains.

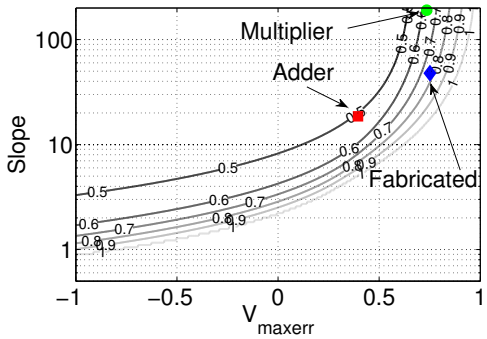
The second significant result shown in Figure 4.7 is that DPSP improves ET^2 on top of speculation, even after accounting for the additional energy overhead of the decoupling queues. The simulated adder has the greatest benefit from decoupling, and ET^2 is improved by an additional 11% on top of SIMD with timing speculation. The benefits are a healthy 12% with the measured multiplier, bringing it below the speculation overhead, but only 3% with the FPGA multiplier. The reason for the small benefit with the FPGA multiplier is that the slope of the error function is very steep, which means that the reduction in effective error rate with decoupling has little impact in terms of energy.

Given the dependence on the error function properties, we evaluate the benefits of timing speculation and DPSP across a large space of error functions. We use our model to compute expected ET^2 for each technique varying both of

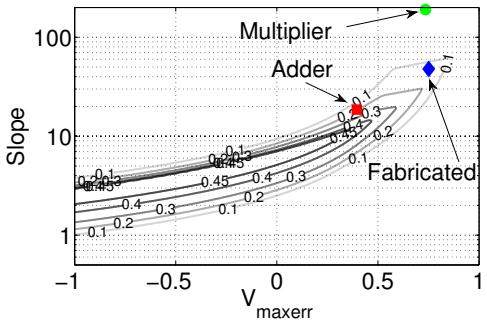
the parameters of our modeled error function Equation 4.4: the relative supply voltage at which error probability is 1 (V_{maxerr}) and the exponential slope of the error probability function (*slope*). Figure 4.8 show the results as contour plots with the "elevation" in the three subfigures corresponding to the ET^2 of SIMD with timing speculation, ET^2 of DPSP relative to no speculation, and the difference between the two (subtraction); all ET^2 is reported relative to



(a) SIMD - lower elevation is better.



(b) DPSP - lower elevation is better.



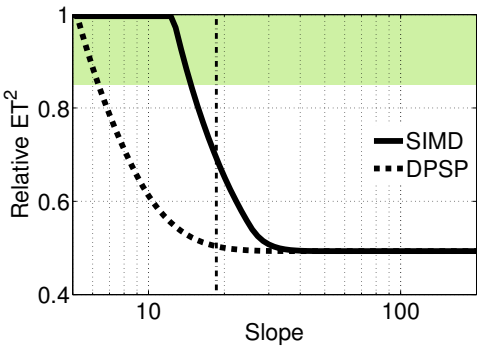
(c) Difference of SIMD and DPSP (higher elevation corresponds to bigger improvement of DPSP over SIMD).

Figure 4.8: Optimal ET^2 with speculation relative to that of SIMD with no speculation.

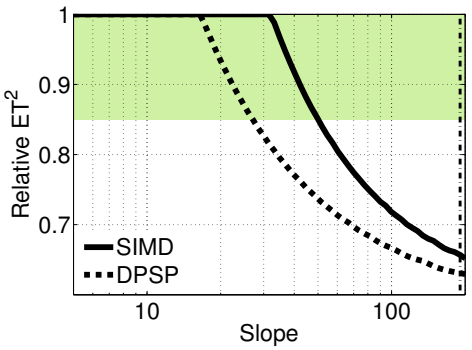
that of SIMD with no speculation. Note that V_{maxerr} is a parameter of the error function model and can be negative; this particularly makes sense for error functions with a gentle slope. We also show three 2D cross sections of the 3D surfaces in Figure 4.9; each subfigure shows the relative ET^2 of speculative SIMD and DPSP with V_{maxerr} set to one of the three evaluated functional units (points indicated in Figure 4.8) and varying slope. Note that we do not show points above relative ET^2 of 1, because at that point speculation should be turned off and DVFS applied.

These results strengthen our insights on when timing speculation and DPSP are beneficial. The steeper the slope, the more potential there is for timing speculation to help. A steep slope indicates that there is a region of V_{dd} in which there are few errors, and decreasing voltage within that region reduces power without significantly impacting performance. The value of V_{maxerr} corresponds to the width of this low-error region. Thus, the greatest benefit from timing speculation is towards the top-left of the surface, with nearly no potential towards the bottom (very gentle slope – many errors in entire range) or right (high probability of error from a high V_{dd}). DPSP provides the most benefit near the diagonal of the surface with potential benefits of above 40% on top of timing speculation with a SIMD pipeline. We confirm our earlier observation that DPSP is most effective when the slope is not extreme. This is clearly shown in the cross-section plots (Figure 4.9). We also see that when V_{maxerr} is larger, timing speculation and the DPSP improvement on top of it, are greater for steeper slopes.

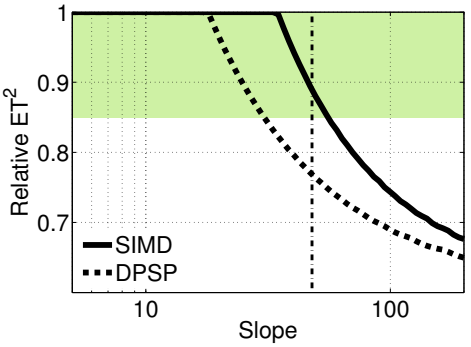
This analysis and insights are valuable because the error probability function is very much design dependent. Rules of thumb and good models are necessary for making informed tradeoff decisions. DPSP, for example, is able to provide benefits even when both the slope and V_{maxerr} are high, which is very encouraging. We also believe that the trends in the underlying fabrication



(a) Adder [16] – $V_{maxerr} = 0.395545254\,V$



(b) Multiplier [16] – $V_{maxerr} = 0.735367316\,V$



(c) Measured fabricated multiplier – $V_{maxerr} = 0.751256185\,V$

Figure 4.9: ET^2 of SIMD with timing speculation and DPSP for varying error function slopes for the three V_{maxerr} values corresponding to the three evaluated circuits; the estimated ET^2 for DVFS with no speculation hardware is shown by the shaded regions. Note the starting point of the y-axis.

technology are likely to lead to error functions that are closer to the sweet spot of DPSP. In an ideal design, $V_{maxerr} = 1$ and the slope is very steep, which result from perfectly balanced paths. Because of process variation and the challenges in balancing complex designs with the large number of optimization options and uncertainties of a modern process, we believe that gentler slopes and non-ideal lower V_{maxerr} values are likely in the future.

4.5 Detailed Microarchitecture Simulation-Based Evaluation

In this section we present analysis based on detailed microarchitecture simulation and validate the model-based results presented above. As explained in Section 4.2, we use the GPGPU-sim [1] detailed microarchitectural GPU simulator, which we modified to support DPSP, timing-speculation, and error injection. We evaluate the set of 12 applications (Table 4.5) included with the GPGPU-sim distribution, which are representative of GPGPU workloads [1]. We do not use the simplified exponential error probability model from Section 4.3 for injecting errors. Instead, we use the error distributions directly and choose the error rate based on the V_{dd} being simulated. We use the cycle-based simulator to estimate the impact of errors and recovery on execution time. To model power, we rely on the technique presented by Hong and Kim [29]. This power model was shown to match well with real hardware, but it's empirical nature requires us to use simulation parameters that are as close as possible to those used to develop the model. To model improvements from

timing speculation, we use Equation 4.7 (Section 4.4.2). When presenting the results we discuss how we account for the fact that only a fraction of overall processor power is reduced because timing speculation is mostly applicable to logic.

Table 4.5: Benchmark applications used in simulation.

Benchmark	Abbr.	Instr. Count [M]	Memory Structures Used			Explicit Synchronization
			Shared	Constant	Texture	
AES Cryptography	AES	28	✓	✓	✓	✓
Breadth First Search	BFS	17				
Molecular Dynamics - Coulombic Potential	CP	126		✓		
Discontinuous Galerkin Time-domain Solver	DG	596	✓		✓	✓
3D Laplace Solver	LPS	82	✓			✓
LIBOR Monte Carlo	LIB	907		✓		
DNA Matching	MUM	77			✓	
Neural Network Digit Recognition	NN	68				
N-Queens Solver	NQU	2	✓			✓
Ray Tracing	RAY	71		✓		✓
MD5 Hashing	STO	134	✓			
Weather Prediction	WP	215				

Figure 4.10 summarizes the optimal ET^2 achieved in simulation for each application when using the three empirical error probability profiles developed in Section 4.3. To find optimal ET^2, we simulated each application with multiple V_{dd} settings (each with its appropriate error rate and power improvement) and chose the best one. We evaluate different configurations of DPSP varying the size of the decoupling queues as well whether the pipeline synchronizes before every memory operation (denoted with DPSP-S*, where * marks the depth of the queue) or just at explicit barriers (DPSP-D*). Note that the ET^2 reported in the figure is under the assumption that the entire processor benefits from reduced V_{dd}. Unfortunately, some structures, such as

the I/O drivers and SRAMs, cannot be improved with timing speculation. The results should be derated by a factor corresponding to the fraction of total processor power of the circuits for which margin can be trimmed. When reporting absolute numbers in the discussion below, we assume that 75% of the power is dissipated by structures amenable to guardband reduction, which is our best estimate based on our previously described power models [29, 40]. Figures 4.11–4.13, however, compare the techniques without this derating factor.

The results show that the analytical model we developed in Section 4.4 correlates well with the overall (average) behavior for both SIMD and DPSP for all error profiles. We also present the behavior of each application across a range of supply voltage values (rather than just the optimal values) in Figures 4.11–4.13. This figure shows that application behavior also matches the model qualitatively. Note that we performed this analysis for all three circuit error profiles to derive the optimal points, but only illustrate the process and sensitivity by showing the detailed behavior for the error profile corresponding to the measured multiplier circuit from our test-chip.

As expected, our model overestimates the ET^2 of timing-speculative SIMD because the natural synchronization points and memory-related stalls of real execution. Surprisingly, however, DPSP with deep queues and minimum synchronization (only on explicit barriers) performs poorly. The reason is that decoupling results in different memory operations executing concurrently than with the SIMD design, resulting in non-coalesced memory accesses and

reduced performance [52]. The DPSP configurations that synchronize before every memory operation do not break software optimization for coalescing. We expect newer GPU architectures, such as NVIDIA's Fermi [54] to be much less sensitive to this coalescing issue because of the introduction of a first-level cache. Unfortunately, the energy model we use does not include a cache and its empirical nature prevents us from adding one. It is also interesting to observe that with all configurations, there is no advantage to deeper queues, and a low-cost 4-deep decoupling queue is sufficient.

Several applications stand out in behaving differently from the average and we discuss each in detail below. BFS (Figure 4.11b) suffers from a very high control divergence rate [1]. The method used by GPUs to handle control divergence in their SIMD pipelines requires synchronization each time multiple control paths reconverge [21]. The high divergence rate of BFS results in many more barrier operations than in a typical application, which limits the effectiveness of decoupling. Note that our mechanism for synchronizing the decoupled lanes stalls the instruction sequencer and thus significantly degrades performance. Notice that the deeper the queues, the higher the penalty for synchronization because the queues are drained before instruction sequencing resumes.

CP, DG, and RAY (Figure 4.11c, Figure 4.11d, and Figure 4.13b) show extreme cases of sensitivity to memory coalescing. Timing speculation with SIMD works quite well and DPSP-S* enables even higher efficiency.

MUM (Figure 4.12c) has unusually low SIMD occupancy with 60% of SIMD instructions (warps in NVIDIA's terminology) having fewer than 5 operations out of a maximum of 32 [1]. As a result, decoupling provides little advantage because the error rate of SIMD is not amplified as much as with other applications. Furthermore, the performance of this application is bound by memory bandwidth, further decreasing the benefits of decoupling. Being memory bandwidth bound, on the other hand, provides a large opportunity for timing speculation to improve efficiency. Even a large number of errors can be tolerated without increasing execution time.

WP (Figure 4.13d) is very sensitive to memory performance and latency. The register file in our GPU configuration is not large enough to support the levels of locality and parallelism required by this application and memory accesses cannot be effectively overlapped with computation [1]. The strong dependence of WP's performance on memory accesses and memory scheduling results in unusual ET^2 behavior as V_{dd} is decreased. SIMD with timing speculation reaches its highest efficiency at relative V_{dd} of 0.9 and further decreases in voltage result in reduced performance and worse ET^2. Decoupling exacerbates the memory scheduling and coalescing issue and is even less effective than lockstep SIMD execution.

4.6 Summary and Future Work

We demonstrate that there is significant potential in extending this circuit/architecture technique to a GPU pipeline. We describe a GPU-specific im-

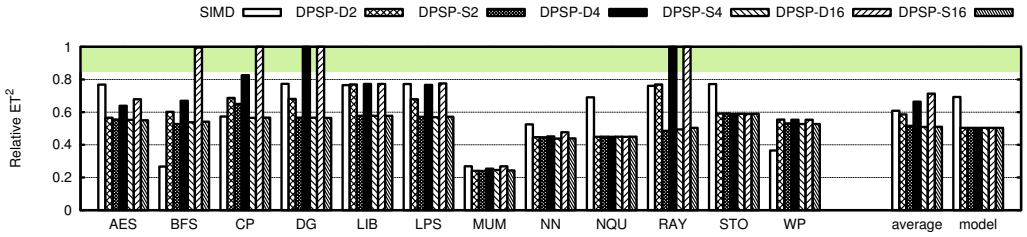
plementation that accounts for the execution model and synchronization/memory-access mechanisms of GPUs. We also describe the use of low overhead decoupling queues to minimally augment the SIMD design, improving both DPSP execution and the tolerance to timing violations. DPSP provides an additional benefit of simplifying the recovery mechanism to enable higher efficiency and performance. Specifically, DPSP enables each SIMD lane to recover from errors independent of other lanes. This implies that any error signal can only be propagated within each simple SIMD lane, which is realizable in a single cycle. Such single-cycle signaling enables stall-based recovery that is most energy and performance efficient. We further evaluate our GPU-based design and show that the peculiarities of its memory access architecture result in the non-intuitive conclusion that a very small degree of decoupling with synchronization for every memory operation yields the best efficiency.

We observe that the general trend in GPU architectures is to reduce the dependence on exact memory optimizations, and therefore expect cache-based GPUs to show greater gains with larger amounts of decoupling. We also describe a potential problem involving high synchronization penalties for highly control divergent applications. While we do not evaluate a solution in this dissertation, recent work on mechanisms to mitigate the negative impact of control divergence (e.g., [21]) are likely to improve DPSP as well.

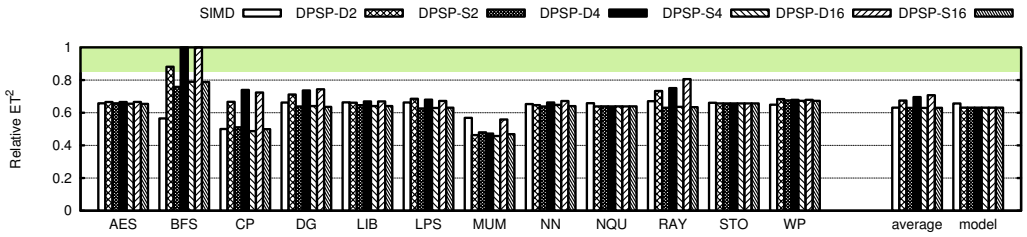
An important contribution of our work is the new detailed models we provide for analyzing timing-speculative SIMD and DPSP designs and the insights we draw from these analyses. We present a novel model for deriving a

timing-speculation induced error probability. This model is based on empirical measurements from a chip fabricated in a modern $45nm$ CMOS process, as well as on previously reported results [16] that have also been used in other related models (VARIUS [64]). We generalize the model using simple and intuitive parameters that allow us to explore the large implementation space. We also develop a new model for the behavior of relative ET^2, and validate both models with detailed cycle-based simulations of a GPU [1].

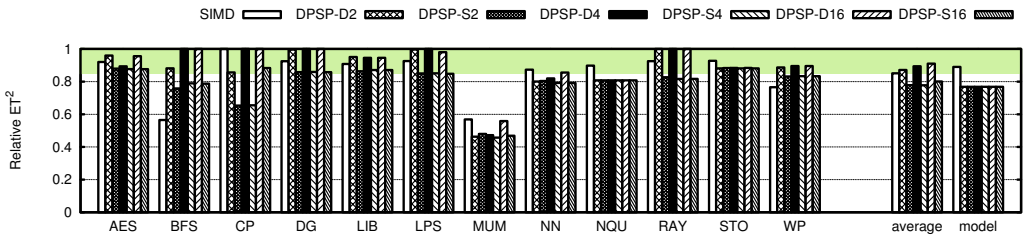
Using this comprehensive evaluation framework leads to interesting insights with respect to timing speculation and SIMD. We note that current trends point to increasing SIMD width for improved efficiency. Based on our analysis, successfully applying timing speculation to such designs requires this new DPSP mechanism, because a naive implementation results in a much higher effective error rate, eliminating much of the advantages provided by operating with reduced margins. We describe how to intuitively interpret the different error probability functions of different circuits on the potential benefits of timing speculation. Finally, we conclude that timing speculation is likely to remain effective in future technologies, as the trends of increasing process variation should lead to a gentler error function slope and a lower relative voltage where the error probability approaches 1. Based on this discussion, we conclude that the average of 10.3% improvement in ET^2 we observe in our evaluation is conservative rather than optimistic.



(a) Adder [16] error profile.

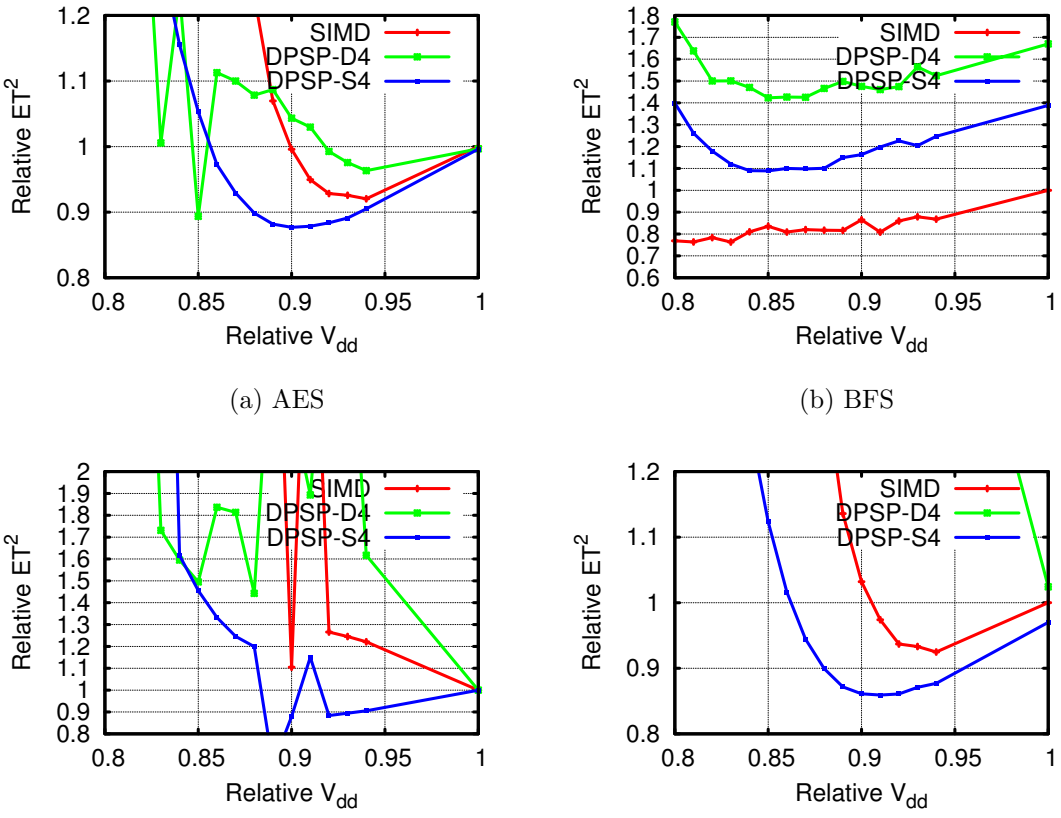


(b) Multiplier [16] error profile.



(c) Measured fabricated multiplier error profile.

Figure 4.10: Optimal ET^2 evaluated with simulation and the analytical model with different configurations DPSP; the estimated ET^2 for DVFS with no speculation hardware is shown by the shaded regions.



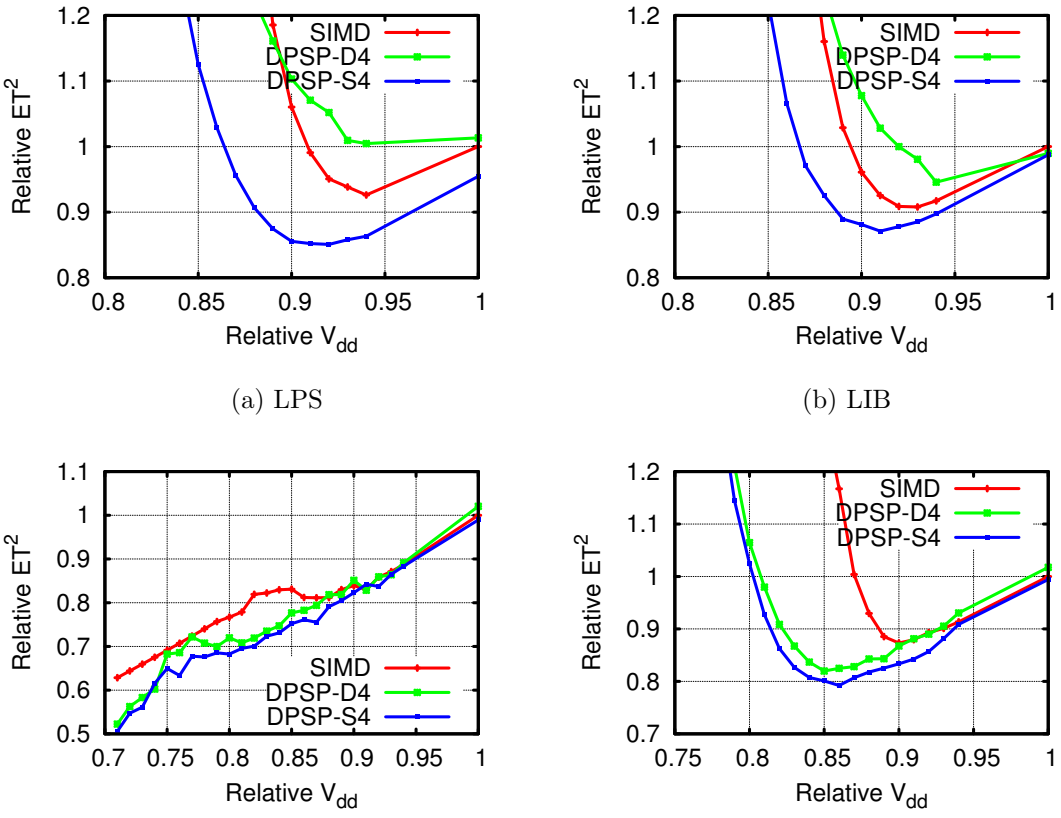
(a) AES

(b) BFS

(c) CP

(d) DG

Figure 4.11: ET^2 as a function of V_{dd} for various benchmarks using the error profile measured for our fabricated multiplier. (AES, BFS, CP, DG)



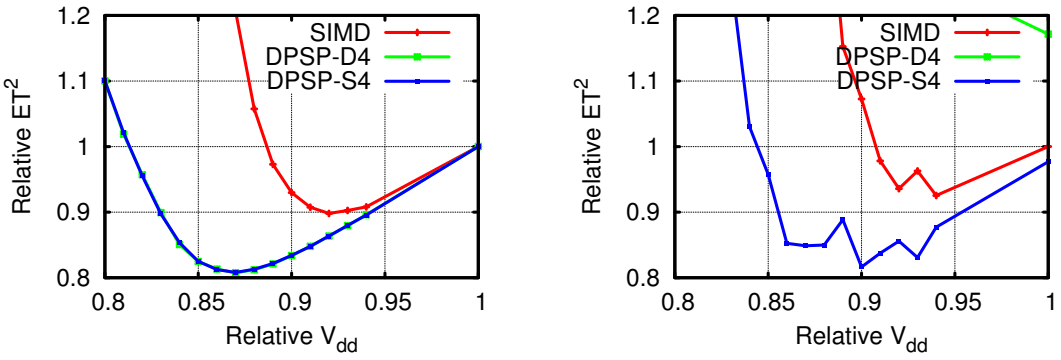
(a) LPS

(b) LIB

(c) MUM

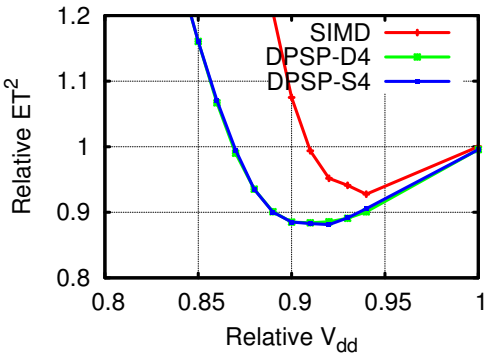
(d) NN

Figure 4.12: ET^2 as a function of V_{dd} for various benchmarks using the error profile measured for our fabricated multiplier. (LPS, LIB, MUM, NN)

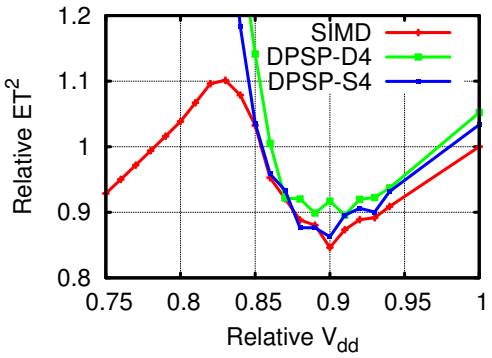


(a) NQU

(b) RAY



(c) STO



(d) WP

Figure 4.13: ET^2 as a function of V_{dd} for various benchmarks using the error profile measured for our fabricated multiplier. (NQU, RAY, STO, WP)

Chapter 5

Diversified Duplex Execution

In Chapter 4 we discussed how SIMD pipelines can be made more efficient by enabling timing speculation using the DPSP approach. In this chapter we focus on the impact and additional complications resulting from the large degree of process variation expected in future technology and when supply voltage is reduced to reduce operating power. We show that the Razor latch [16], which was also used as the underlying timing violation detection circuit within DPSP fails when process variation is large. We therefore propose, evaluate, and discuss a novel approach for detecting timing violations that works correctly even under extreme variation.

The Razor flip-flop (see Section 3.3.1.2 on page 24) is perhaps the most well known mechanism that enables timing or voltage speculation. Part of the motivation for the Razor approach is that it improves efficiency in the face of process variation. Although Razor does reduce some of the margins associated with variation, it has two significant disadvantages. First, Razor cannot operate correctly when process variation is extreme because it places strict timing constraints on the longest and shortest delay paths. Second, Razor requires area and power overhead even when no speculation is performed, such as when

operating a particularly good chip or when operating with parameters in which variability is small.

By design, Razor requires all path delays to be between 1/2 and 3/2 of the operating cycle time. This is because outside of this region errors may either slip undetected or short paths will lead to a very high false alarm rate as data from the next cycle overtakes slow paths in the current cycle. Delay variability, which is already above 50% (see Figure 3.2 on page 22) [30] results in a wider range of delay uncertainty, however, thereby making the double sampling error detection approach infeasible.

Moreover, Razor implies hardware overhead, including the additional shadow latch for each protected flip-flop and buffering logic to guarantee the minimum-path delays that are at least half of the cycle time. This extra hardware has to be part of the design to allow potential speculations resulting in area and power overheads regardless of actual speculation usage. Thus, it is not suitable for systems that are usually at their nominal operating point and only occasionally require speculating due to, for example, a drastic environmental change (temperature increase) or ultra dynamic voltage-frequency scaling (DVFS) to near-threshold supply voltage [7].

To address these two issues, we propose a new timing violation detector based on *diverse duplex execution* (DDE). Using diverse designs of the same (execution) unit, such that the input exercising the critical path in design A will not exercise the critical path in design B and vice versa, a timing violation will result in an output mismatch (Figure 5.1). Another reason for a mismatch

can be a soft error. Thus this diversified duplex approach inherently provides soft error protection in addition to intermittent errors protection.

In modern GPUs, like the Tesla-C2050, the register file and the memory are protected using error checking and correcting (ECC) codes [55], leaving the front-end (instruction sequencer) and arithmetic logic units (ALUs) vulnerable to both intermittent and soft errors. Since the front-end is shared among multiple lanes, the overhead for protecting it against errors is amortized among the lanes. Therefore, in this research we prefer to focus on evaluating the diversified duplex execution only for protecting ALUs.

The diversified duplex approach allows building a dynamically configurable SIMD architecture (Figure 5.2) that will operate in duplex mode only when speculation is desired. Such an architecture requires deploying diversified units that in "speculation mode" will be dynamically reconfigured to operate in duplex mode. However, in normal mode, when no speculation is allowed,

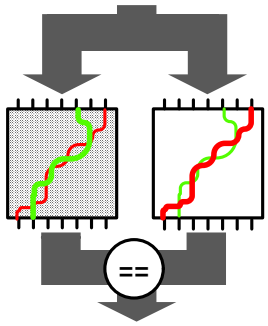


Figure 5.1: Diverse designs of the same unit having different critical paths (marked in bold), fed with the same inputs. The outputs are matched to detect intermittent and soft errors.

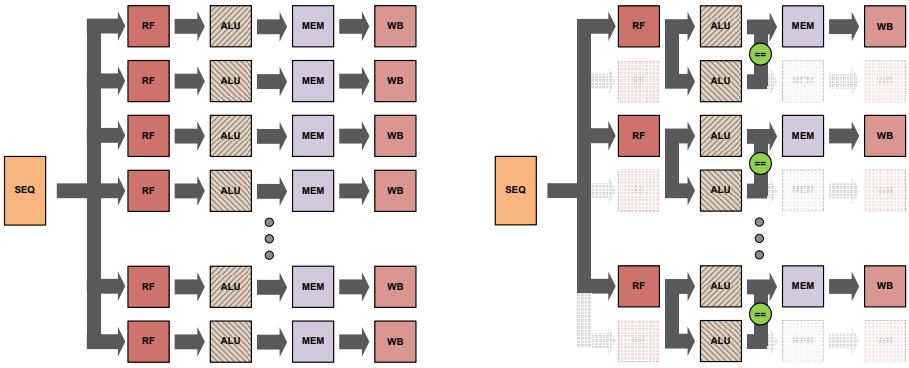


Figure 5.2: Dynamically reconfigurable SIMD architecture based on diversified duplex error detection. Adjacent ALUs are using diverse designs. (a) in normal operation mode; (b) in speculation mode.

there will be almost no penalty. As part of this research we will evaluate the overhead due to diversity requirement and the error detection capability of the diversified duplex approach.

Finally, we propose to apply DDE to systems that use a dual modular redundancy mode for higher reliability. Using the diversified duplex error detector, such systems can speculatively reduce their supply voltage V_{dd} and thus reduce energy consumption.

The rest of this chapter is organized as follows: We discuss the properties of diversity in Section 5.1 and define metrics for its evaluation in Section 5.2. Then, in Section 5.3, we elaborate on different approaches diversity can be obtained, following a discussion of the recovery mechanism in Section 5.4. In Section 5.5, we evaluate the operation of DDE under process variation. Then, in Section 5.6 and Section 5.7, we present examples of DDE

using a 16-bit *carry look-ahead adder* (CLA) and a simple logic function. Finally, we summarize and discuss future research in Section 5.8

5.1 Quantifying Diversity

To allow DDE, the two designs should provide sufficient diversity. In this section we develop a set of metrics to evaluate the diversity of designs. We also discuss a previously published diversity metric [46].

The main concern due to process variation that we are addressing with DDE is computation delay that may not meet timing constraints imposed by cycle time. Therefore, to evaluate the amount of diversity, we propose using a 2D plot where the axes correspond to the delay of the two designs as shown in Figure 5.3. Each point on this plot represents an input vector showing its computation delay requirements by both designs as the X, Y coordinates. Such plots can be optionally drawn for each of the outputs separately, allowing a more fine-grained diversity evaluation. To better understand diversity requirements we present a few representative illustrative examples in Figure 5.3.

We begin with the best case of diversity, shown in Figure 5.3a. The input vectors that show the longest delay with design A (bottom right), actually present short delay with design B. Vice versa, the inputs that present long delays with design B (top left) show short delays with design A. In addition, there is a group of inputs that show short delay times for both designs (bottom left). Given such distribution of delays, operating with cycle-time marked by the blue lines can ensure fully correct execution since the inputs that may not

meet the reduced timing constraints (marked by the blue lines) with one of the designs are guaranteed to do so with the second design.

The blue lines (Figure 5.3a) divide the plot into 3 regions. The bottom left region is the error-free region. Inputs in the error-free region meet the reduced timing constraints by both designs and therefore do not result in a mismatch. The non-detectable error region (empty in this example) is on the top right. Inputs in this region might fail to satisfy the reduced timing requirements in both designs, leaving a chance for such an event go undetected. Given our assumption of correctness requirement, the non-detectable error region must be empty of input vectors. In other words, timing constraints can be reduced as long as there are no inputs that fall into the non-detectable region. Finally, the detectable error regions are located at the top left and bottom right. These regions represent inputs that might fail the timing constraints with one of the designs (at most). In such a case a recovery will be required. Thus, highly populated detectable regions may cause significant performance degradation due to the need for recovery. Note that this plot does not present the frequency of occurrence of each input. However, for simplicity, in this stage we can assume that the occurrence of the input vectors is uniformly distributed.

A case of extremely low diversity is shown in Figure 5.3b. All the delays are concentrated around the equity line (diagonal). The main issue in this case is that there are input vectors occupying the right top corner of the plot, leaving no room to reduce to timing constraints. Any reduction in timing

requirements will result in having input vectors in the non-detectable region, contradicting the correctness requirement.

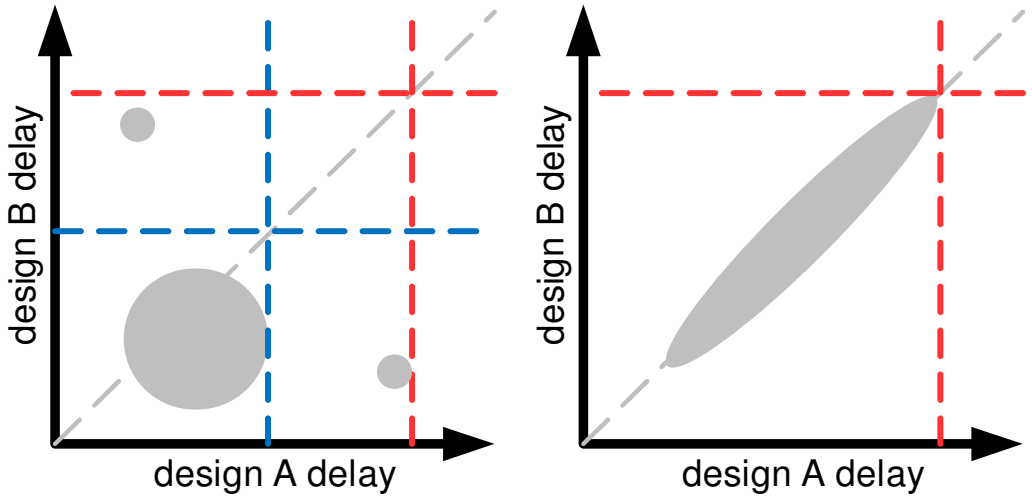
Figure 5.3c shows a case with high diversity (the points are spread). Neither design has input vectors that require a long computation delay, resulting in a relatively small potential for reduction in timing constraints.

Finally, Figure 5.3d shows a case of asymmetric designs. The majority of the input vectors in this case are below (or can be above) the equity line, meaning that the delay presented by design B is always shorter than the delay presented by design A. Although according to the definition beforehand, this plot has no the potential to reduce timing constraints, the same result can be achieved by using design B only (or two instances of design B if required for redundancy).

5.2 Metrics

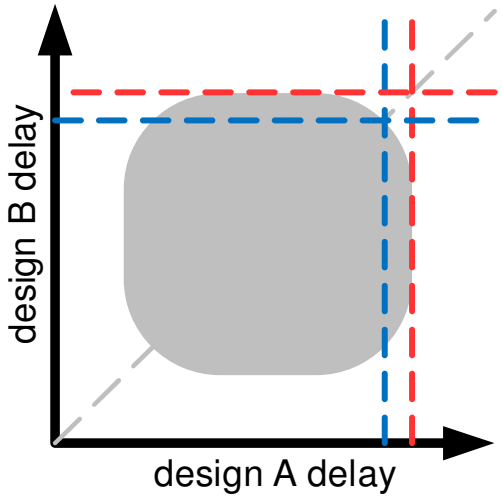
Following our observations in Section 5.1, we develop a set of metrics to describe the diversity between two designs for the purpose of timing speculation using DDE. The first metric is δ_{max}, which corresponds to the maximum possible reduction of timing constraints without the introduction of non-detectable errors.

The second metric is $\epsilon(\delta)$, which represents the probability of a detectable error given timing constraints reduction of δ. Let $D_\alpha(v)$ and $D_\beta(v)$ be the computation delay for the input vector v shown by designs A and B,

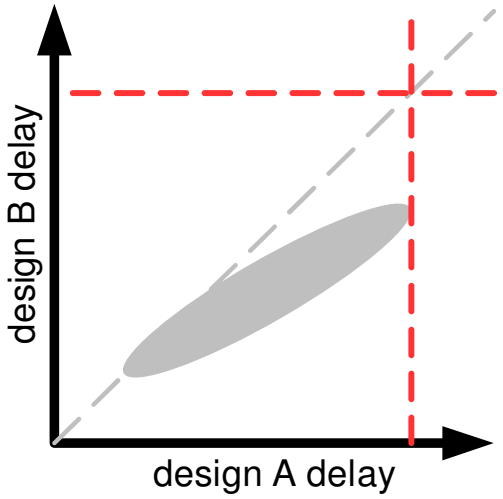


(a) Best case

(b) Low or no diversty



(c) Low timing constraints reduction potential



(d) Asymmetric designs

Figure 5.3: Various scenarios of computation delay diversity. Each point in the plots corresponds to an input vector showing required computation delay by each of the designs. Equity line (diagonal) is presented in light grey. The worst (global for both designs) case delay is shown by the red lines. The blue lines show the minimum timing constraint ensuring fully correct execution.

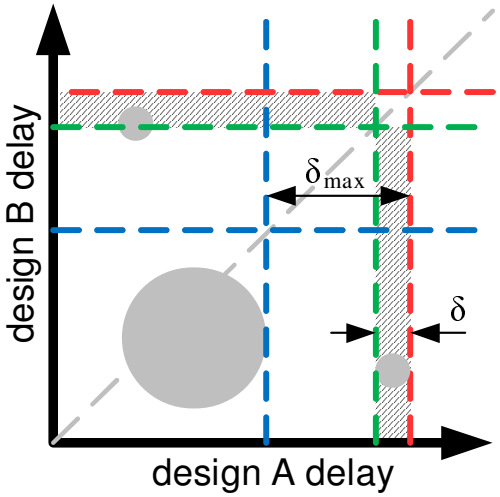


Figure 5.4: Detectable errors due to timing constraints reduction. Worst-case timing (initial conditions) marked in red. Maximum timing constraint reduction, δ_{max}, is marked in blue. Green lines show a reduction in timing constraints by δ. Shaded area represents the detectable region given a reduction in timing constraints of δ.

respectively, and let $P(v)$ be the probability of input v to occur and T be the initial timing constraint; then:

$$\epsilon(\delta) = \sum_{v \in \text{detectable}} P(v) = \sum_{\max\left(D_\alpha(v), D_\beta(v)\right) > T - \delta} P(v) \tag{5.1}$$

Additional metrics include the overheads in energy due to the diversity of both designs A and B. Let E_α and E_β be the average energy consumptions of designs A and B, respectively, and let E_{opt} be the average consumption of the (energy-) optimal design implementing the same functionality. Then the energy overheads can be defined as $\rho_x = \frac{E_x}{E_{opt}}$ for $x \in \{A, B\}$.

A different diversity metric was developed by Mitra et al. [46] in the context of using the diversity to improve the reliability of systems implementing modular redundancy. In that work, the probability of an undetected fault is used as a metric. However, since in our case we assume the 100% correctness requirement, this metric should be a constant zero and therefore is not suitable in our context.

5.3 Achieving Diversity

So far we have discussed how to evaluate the amount of diversity presented by two designs. In this section, we discuss the possible approaches to achieve the desired diversity. We distinguish between different levels at which the diversity can be achieved.

5.3.1 Algorithmic Level

At the algorithmic level, diversity can be achieved by using two instances of the exact same design in different manners. For example, in the case of an unsigned adder, the second design can use a complement of the inputs relaying on the fact that $\mathrm{ADD}(A, B) = \overline{\mathrm{ADD}(\overline{A}, \overline{B})}$. Due to its nature, this approach is extremely sensitive and should be tailored to the specific functionality of the design.

Our experiments with a 16-bit *carry look ahead* (CLA) adder showed a decent diversity of computation delays. However, there are some input vectors

that result in similar high computation delays in both designs leading to $\delta_{max} \approx 0$.

5.3.2 Architecture Level

Diversity can be achieved by using different implementation architectures. For example, in the case of adders, the implementation can differ (CLS, Kogge-Stone, Ladner-Fischer etc). Although this approach is more generic than the algorithmic level, it is still limited and not universally applicable.

Our experiments with 16-bit adders showed asymmetric results (see Section 5.1). One of the adders showed noticeably better performance compared to the other, resulting in $\delta_{max} = 0$.

5.3.3 Logic Level

The same logic terms can be calculated in various ways. For example, using distribution or De Morgan laws, logic equations can be modified while being fully equivalent. These modifications may eliminate some critical paths while introducing others and thus provide the desired diversity.

We present the results obtained from our experiments with logic level diversity using a 16-bit CLA in Section 5.6.

5.3.4 Circuit Level

We distinguish between circuit and logic levels mostly for semantics and presentation purposes. At the circuit level, we consider changing the sizes

of the gates or transistors or even using different gate cells. Moreover, while operators like NOR, NAND, etc., are commutative, the gates implementing these operators have slightly different response time properties due to changes in different inputs. Therefore, simply swapping the inputs will improve some paths at the expense of others and provide diversity.

We experimented with swapping all the inputs in a 16-bit CLA; however, the amount of diversity was not sufficient. To experiment with gate sizing, we tried an extremely simple logic function implemented with a few gates through an exhaustive search over different sizes. Each combination was simulated using SPICE. The results of this sizing experiment, however, were also not sufficiently diverse, probably due to the simplicity of the logic. Experiments with multi-Vt design [49], however, provided sufficient diversity. We present this experiment in Section 5.7 and use it to demonstrate that DDE is applicable even with high process variation.

5.3.5 Layout Level

The lowest level at which diversity can be achieved is the physical layout level. By biasing layout optimization decisions, different paths can be improved (at the expense of other paths). For example, by changing the placement policy of metal allocation the same design can be laid out differently, providing diversity.

We did not perform any experiments to achieve diversity at the layout level.

5.4 Recovery

To complete the description of the proposed mechanism, in this section we describe the recovery process in case of error (mismatch) detection. Both the flush and stall approaches (see Section 3.3.2 on page 30) are suitable. Use of the flush and replay approach may lead to performance degradation as a result of the replay process. On the other hand, to implement recovery by stall, the stall signal should be generated and propagated before the next cycle occurrence.

We leave the detailed evaluation of the overheads due to stall recovery as well as its comparison to flush and replay for future work.

5.5 Process Variation

Until now, we primarily discussed only the computation delay differences presented by a design due to different input vectors. In this section we extend the analysis of DDE to the presence of process variation. Chip designers striving to maintain high yield usually consider $\mu + 3\sigma$ as the worst acceptable delay of circuit, where μ is the average delay and σ is the standard deviation. We follow this convention and add this variation to the diversity plots. This added variation results in a slight shift of the points on the diversity plot towards the top right, as shown in Figure 5.5.

As we show for illustrative purposes in Figure 5.5, the magnitudes of σ_α and σ_β that correspond to a specific input applied to designs A and B re-

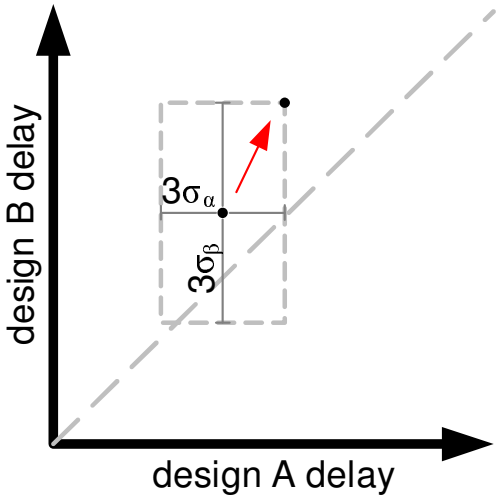


Figure 5.5: Delay shown by each of the designs should be extended by 3σ. A new point corresponding to the input vector is shown in the right top corner of the rectangle.

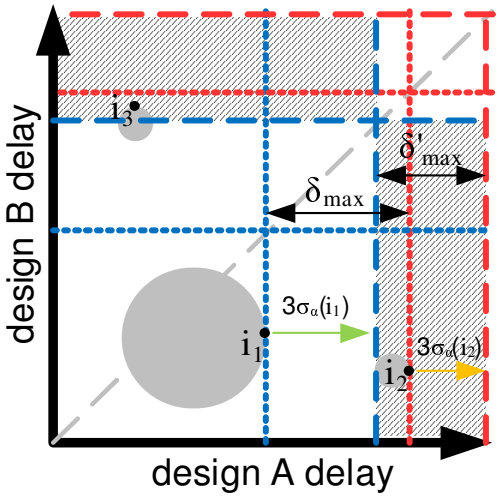


Figure 5.6: δ'_{max} represents the diversity potential adjusted to process variation. δ_{max} is the initial diversity potential without taking process variation into account.

spectively depend on the actual designs; i.e., for each specific input vector, the impact of process variation is different in each design ($3\sigma_\alpha(i_1) \neq 3\sigma_\beta(i_1)$). Additionally, as depicted in Figure 5.6, the standard deviation shown by each of the designs varies between different inputs ($3\sigma_\alpha(i_1) \neq 3\sigma_\alpha(i_2)$). Our simulations show a linear dependency between σ and μ for both of the designs we tested ($\sigma_{alpha} \propto \mu$ and $\sigma_{beta} \propto \mu$). Therefore, given diversity of δ_{max}, which is determined based on the worst-case inputs i_1 and i_2, we derive the magnitude of the diversity adjusted to process variation, δ'_{max} as:

$$\begin{aligned}
\delta'_{max} &= (\mu(i_2) + 3\sigma_\alpha(i_2)) - (\mu(i_1) + 3\sigma_\alpha(i_1)) \\
&= (\mu(i_2) - \mu(i_1)) + (3\sigma_\alpha(i_2) - 3\sigma_\alpha(i_1)) \\
&= \delta_{max} + (3\sigma_\alpha(i_2) - 3\sigma_\alpha(i_1)) > \delta_{max}
\end{aligned} \tag{5.2}$$

Equation 5.2 may not hold if the two designs respond differently to process variation due to their implementation properties (transistors sizes etc.). For example, in case of a severe difference in response to process variation by the two designs, adjusting to process variation will result in an asymmetric case and loss of diversity as shown in Figure 5.3d.

As discussed beforehand, Razor can operate only while the slowest and fastest delays are between $\frac{1}{2}T$ and $\frac{3}{2}T$, i.e., while $\frac{T_{max}}{T_{min}} < 3$. Therefore, when $\sigma > \frac{\mu}{6}$, $T_{max} = \mu + 3\sigma > \frac{3}{2}\mu$ and $T_{min} = \mu - 3\sigma < \frac{1}{2}\mu$, resulting in $\frac{T_{max}}{T_{min}} > 3$, Razor can no longer be used to detect errors and improve efficiency. In fact, it is quite possible that Razor stops functioning even with less variation because

it is infeasible to buffer all paths to be precisely of equal average delay and because no practical circuit can operate all the way to the $\frac{1}{2}T$ or $\frac{3}{2}T$ boundaries. On the other hand, as we show above there are no such inherent restrictions with DDE, which continues to detect timing-violation errors.

5.6 Example of Obtaining Diversity at the Logic Level

To demonstrate an example of diversity we use a 16-bit carry lookahead adder (CLA) synthesized using the Synopsys Design Compiler with the FreePDK [70] 45nm library. To achieve diversity, the logic is synthesized using different design compiler optimization settings. In the first approach, the logic is fully flattened, removing the boundaries between different subunits. This way, the compiler is able to perform optimizations across subunit boundaries. Then, to achieve diverse design, we avoid flattening the logic, maintaining the subunit boundaries, enforcing local optimization at the subunit level only.

Figure 5.7 depicts the computation delays of two diverse 16-bit CLA designs synthesized using the approach described above. Using SPICE, we simulate 10000 random input vectors applied to both designs and measure their delays. Each point represents a single input vector, and its coordinates (x, y) correspond to the delays measured with designs A and B, respectively. The maximum delays observed are $443ps$ for design A and $436ps$ for design B. Note that design B is the optimal design achieved using our design flow; design B shows the shortest maximum delay of all designs, but consumes more energy and is more sensitive to process variation when compared to design A..

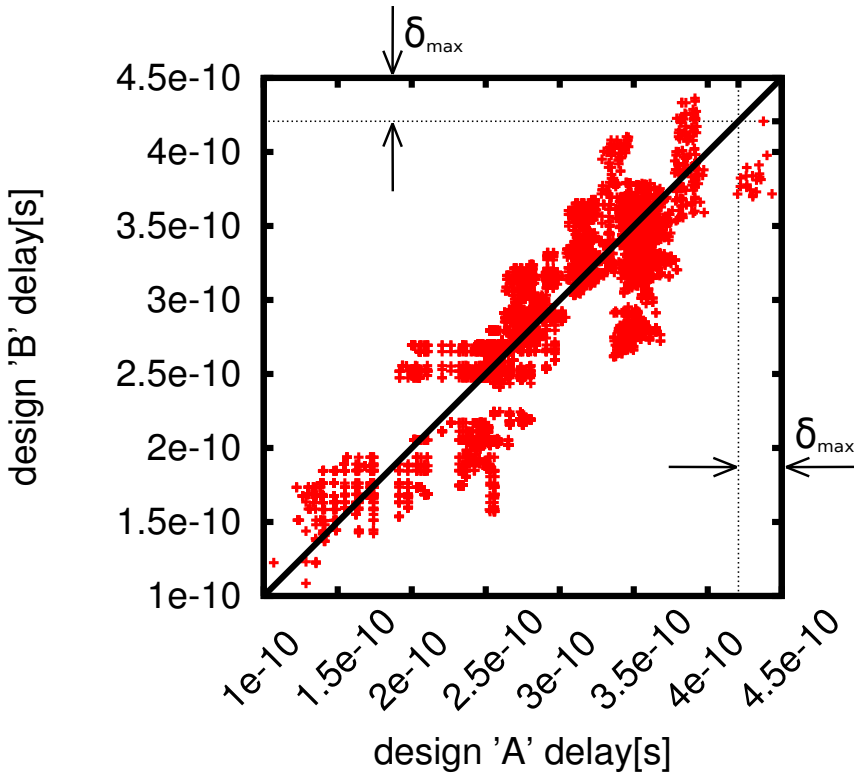


Figure 5.7: Computation delays of two diverse 16-bit CLA adder designs executing 10000 random input vectors. δ_{max} shows the potential cycle time reduction.

The dashed lines show the minimum required delay T_{min} to detect any error, which is $420.7ps$ in this case. Figure 5.8 shows the distribution of computation delays for both designs.

Let $D^X_{i,j}$ be the worst case delay of the path from input i to output j in design X; then, the minimum required delay to detect any error T_{min} is derived in Equation 5.3. Comparing the delays of the two designs for each input/output pair, we determine the minimum required delay for at least one

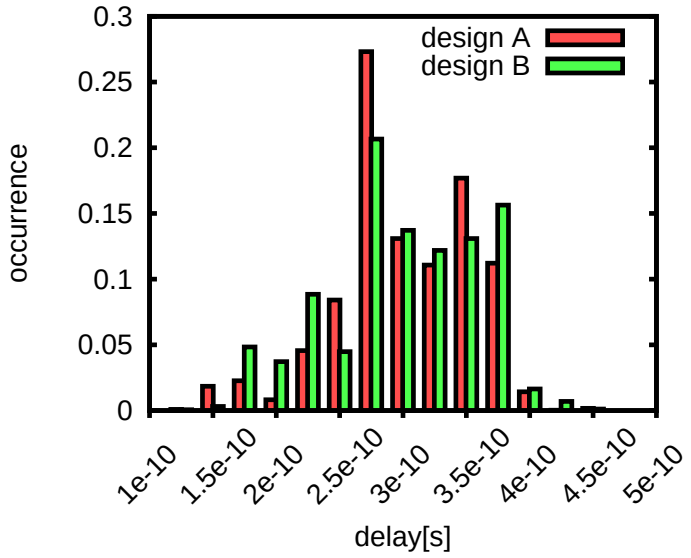


Figure 5.8: Computation delay distributions of two diverse 16-bit CLA adder designs executing 10000 random input vectors.

of the designs to achieve correct output. Then, T_{min} is set to guarantee a correct result for all input/output pairs.

$$T_{min} = \max_{\forall i,j} \left(\min \left(D^A_{i,j}, D^B_{i,j} \right) \right) \tag{5.3}$$

Finally, Figure 5.9 summarizes the error rates for detected and undetected errors as a function of cycle time reduction using diversified decoupled execution. It shows that cycle time can be reduced to $420.7ps$ with no undetectable errors occurring. The probability of an error at that point is 0.42%. Needless to say, all the errors are detectable by construction due to our correctness requirement.

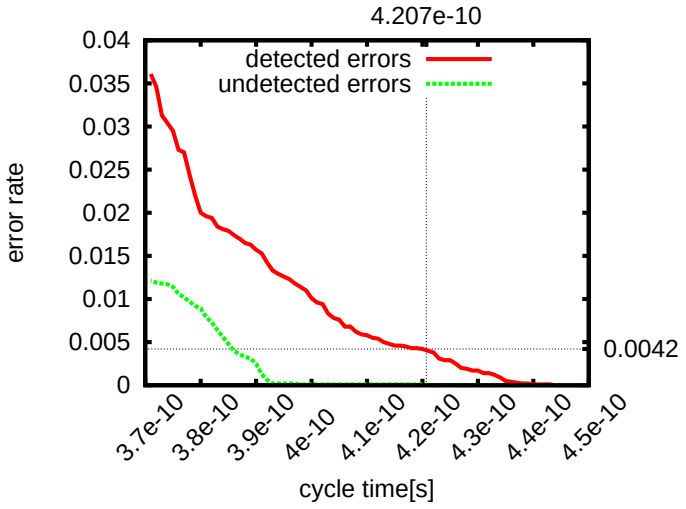


Figure 5.9: Error rates as function of cycle reduction.

5.6.1 Process Variation

The simulation results presented so far assumed no process variation. In this section, we evaluate the impact of process variation on DDE. We use the Monte Carlo method to simulate the random component of the process variation in SPICE.

Due to long simulation times involved with this approach, we limit our experiments to a set of the 10 longest paths in both designs assuming that these will constrain δ_{max}. Figure 5.10 shows the results for the 10 worst paths where the average measurement for each path is surrounded by 3σ margins to illustrate the worst scenario due to process variation.

Figure 5.11 summarizes the potential possible timing constraint reduction δ_{max} using DDE. To compare data obtained with various supply volt-

ages resulting in different initial timing constraints, we choose to present $\frac{\delta_{max}}{\text{timing contrain}}$, which has a common scale.

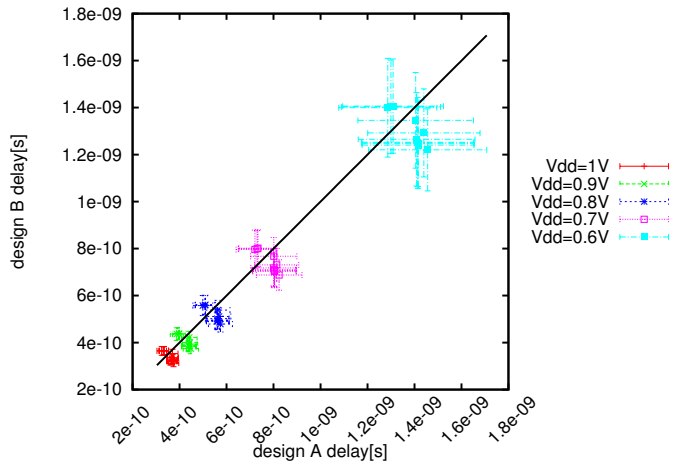


Figure 5.10: Diversity results for worst-case paths of designs A and B in the presence of process variation with varying supply voltage.

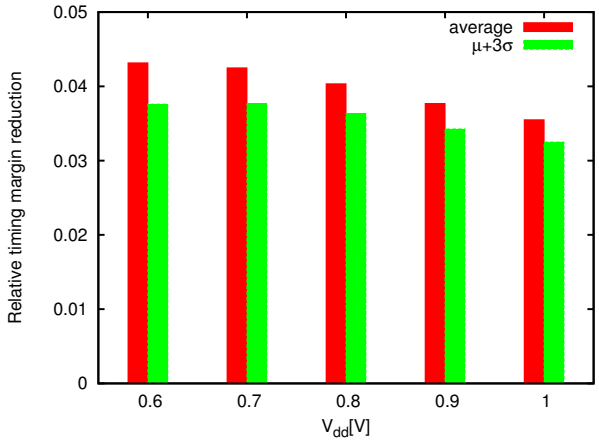


Figure 5.11: Timing margin reduction relative to the fastest design of the two (design B). Results are shown for the presence of process variation as well as based on the average values that represent lack of process variation.

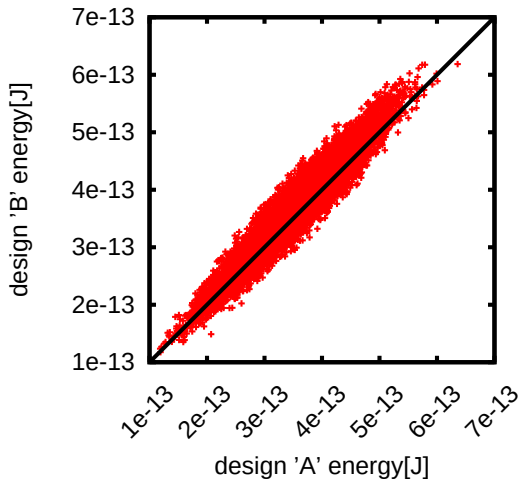


Figure 5.12: Energy consumption of two diverse 16-bit CLA adder designs executing 10000 random input vectors.

5.6.2 Diversity Energy Consumption Overheads

Figure 5.12 shows the energy consumption of designs A and B. Each point represents a single input vector and its coordinates (x, y) correspond to the energy consumption measured for designs A and B, respectively. Although the plot is mostly concentrated on the equilibrium diagonal, we observe slight higher energy consumption with design B. The average energy consumption is $351fJ$ with design A and $365fJ$ with design B, a 4% difference. Part of our future research is to better understand the tradeoff between the level of diversity and the energy overhead.

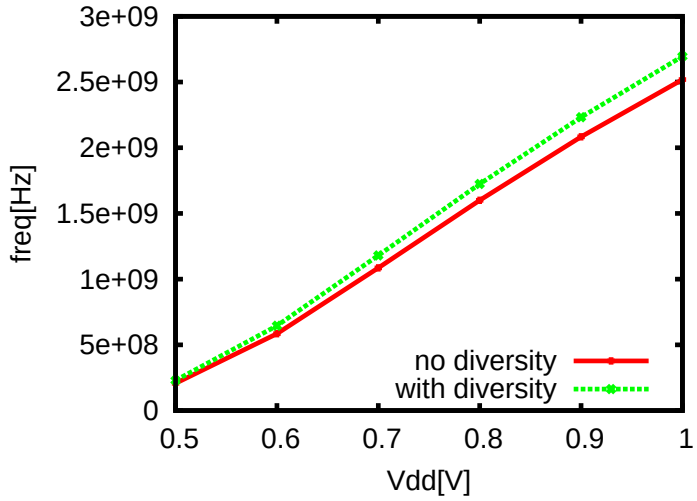


Figure 5.13: Performance improvement of high reliability SIMD architecture using diversified execution.

5.6.3 Usage in a High-Reliability SIMD Architecture

The proposed error detector can also be used in systems implementing dual modular redundancy for single event upset (SEU) protection. By introducing diversity to the design of the redundant units, we can speculatively reduce the supply voltage and improve energy efficiency (Figure 5.14) or reduce cycle time margins and increase the frequency (Figure 5.13).

5.7 Example of Obtaining Diversity at the Circuit Level

At the circuit level, we demonstrate obtaining diversity using a multi-Vt design [49]. Multi-Vt design (also referred to as Multi-threshold CMOS - MTCMOS) is an approach of using transistors with different V_{th} within the

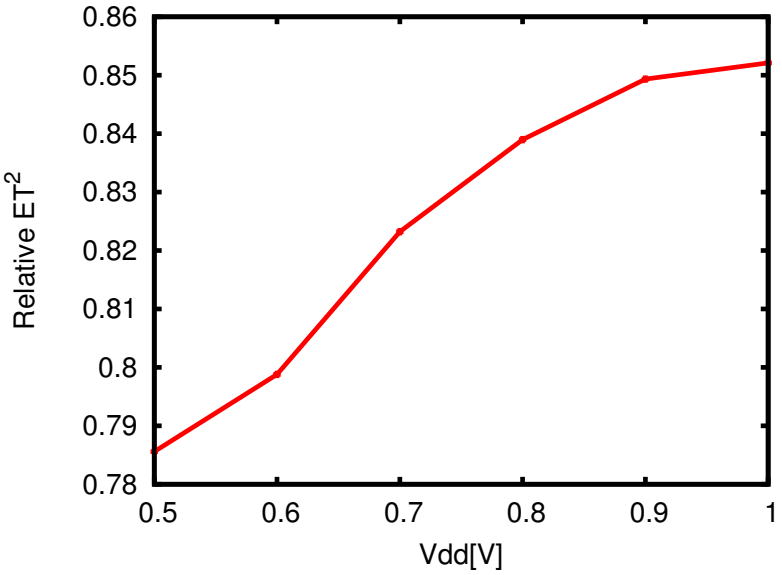
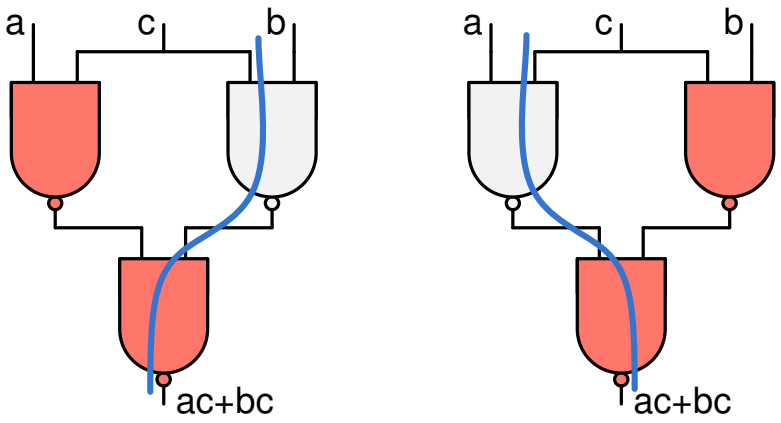


Figure 5.14: The ET^2 of a high-reliability SIMD architecture that uses diversified execution relative to a baseline SIMD architecture with no reliability features.

same circuit, typically with some of the transistors using a low V_{th} and others using a high V_{th}. Transistors with low V_{th} (referred to as LVT) switch faster but also result in higher leakage overhead compared to transistors with high V_{th} (referred to as HVT) [53, 62]. This basic principle enbables trading off the switching delay with the energy consumption of a circuit.

The example below shows how diversity can be achieved with a multi-Vt design. We evaluate a simple circuit that consists of 3 NAND gates, which we implement in two different ways. The first design uses an HVT gate for one pair of inputs and LVT for the other two gates, while the second design uses the HVT gate for the second input pair (Figure 5.15). By placing an

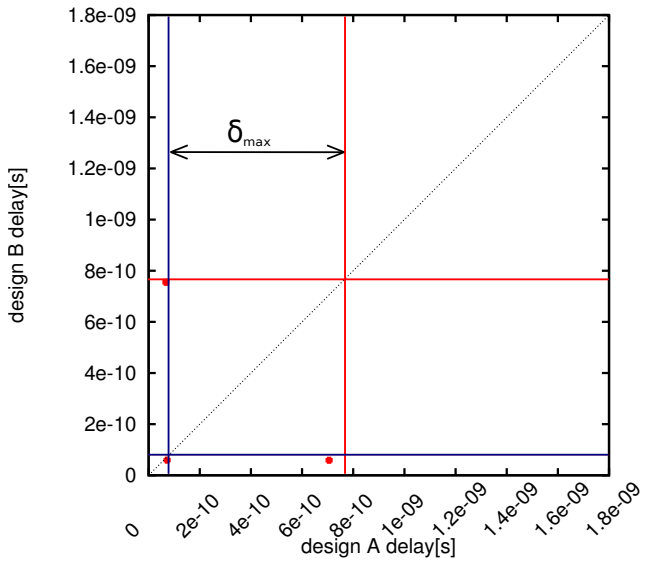


(a) (b)

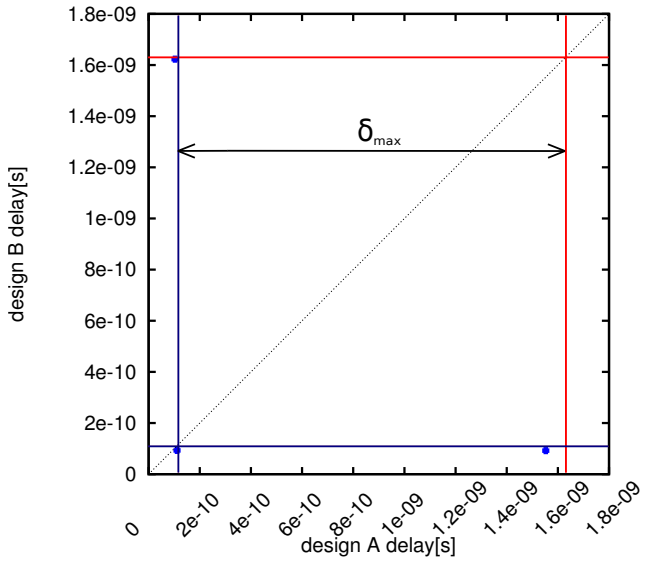
Figure 5.15: Two versions of a simple circuit of 3-NAND gates diversified using multi-Vt design to have different critical paths; gates marked in red represent gates with LVT transistors while other gates use HVT transistors.

HVT gate for different inputs, the critical path of the two circuits differs for different inputs. We simulate both designs in SPICE using the FreePDK [70] 45nm library and predictive technology models [78] for multi-Vt transistors.

Figure 5.16 shows the diversity plots obtained with and without process variation. Process variation is simulated using $\frac{\sigma_{th}}{V_{th}} = 40\%$ [30]. As expected (Section 5.5), the maximum timing constraint reduction (δ_{max}) is higher with the introduction of process variation. We also observe that process variation doubles the computation delays for some inputs in both designs. This implies that the gap between the worst-case instance ($\mu + 3\sigma$) and the best-case instance ($\mu - 3\sigma$) exceeds the min/max delay ratio constraints needed for Razor because no static buffering will solve this problem for a single path.



(a)



(b)

Figure 5.16: Computation delays of two diversified instances (Figure 5.15) simulated at $V_{dd} = 0.6\,V$: with no process variation (a) and with process variation (b); δ_{max} shows the potential cycle time reduction.

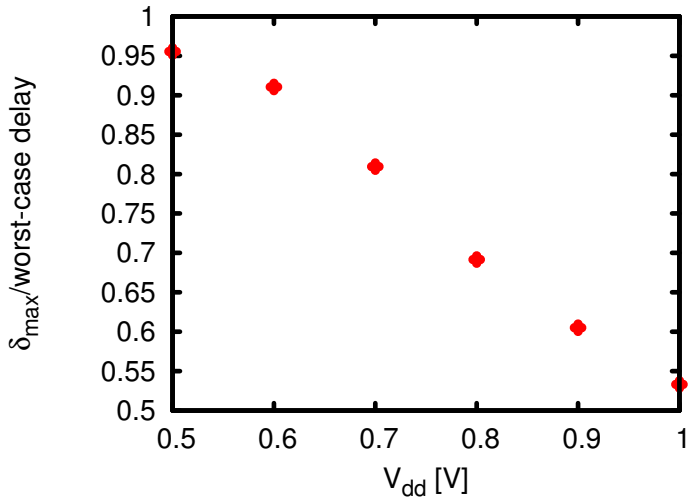


Figure 5.17: Timing constraint reduction potential (δ_{max}) of the diversified design (Figure 5.15) as a function of supply voltage V_{dd}.

To further evaluate this relation between process variation and potential timing constraint reduction (δ_{max}), we repeat the experiment with varying supply voltage and measure δ_{max} in the presence of process variation. Reduced supply voltage results in a higher impact of process variation on switching delay and thus imitates the future technologies which are projected to suffer from higher process variation. Our results (Figure 5.17) show a growth in δ_{max} with V_{dd} reduction indicating that not only is DDE immune to process variation, DDR actually becomes more attractive with the higher process variation expected in future technologies. Furthermore, DDE enables efficient reconfiguration and can operate with high efficiency in both super-threshold and near-threshold regimes.

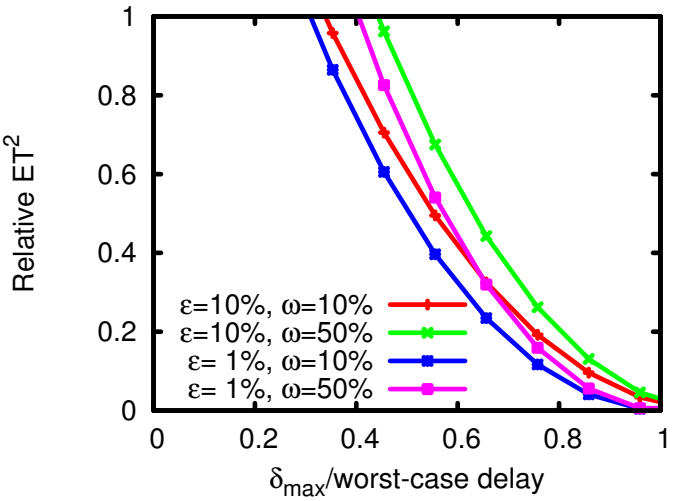


Figure 5.18: Relative ET^2 due to DDE as a function of potential timing constraint reduction (δ_{max}) assuming ϵ error probability and ω overhead due to DDE.

Finally, we evaluate the ET^2 reduction as a function of δ_{max} assuming different error probabilities (ϵ) and overheads due to diversity (ω) (Figure 5.18). We use the model for ET^2 presented in Chapter 4 with a slight adjustment to account for the overhead of DDR. A factor of $2(1+\omega)\times$ is added, where the factor of two corresponds to using two instances of a design with DDE and ω corresponds to potential inefficiencies introduced to one of the designs to obtain diversity. Thus, for example, $\omega = 10\%$ corresponds to a configuration where each of the designs (A and B) consumes 10% more energy compared to the optimal design.

Following the range of δ_{max} observed in Figure 5.17, we conclude that ET^2 can be significantly reduced using DDE, despite the fact that it requires replication and multiple designs. Moreover, since δ_{max} increases with process variation, we expect DDE to present more aggressive ET^2 savings in the presence of higher process variation.

5.8 Summary and Future Work

In this chapter we present diversified duplex execution (DDE), a mechanism for detecting errors resulting from speculating that timing constraints will be met that works even when process variation is high. Unlike the existing techniques, the proposed technique allows trimming of the guardbands for both systematic and random components of process variability as well as input dependent delay fluctuations in the presence of severe process variation. In addition, in the context of a massively parallel architecture, the overhead of the proposed technique is minor when not in use.

The success of DDE relies on achieving the required degree of diversity between two designs. We discuss different ways in different system levels in which diversity can be achieved. While we show significant diversity achieved only in the logic circuits level, we believe obtaining diversity in other levels is possible as well, but leave such techniques for future work as they are better suited for research on design automation methods.

We also evaluate the relation between the degree of diversity achieved the potential efficiency gains of DDE. We show that significant improvements can be attained when achieved diversity is high.

Chapter 6

Conclusions and Future Research Directions

This dissertation focuses on improving the energy-efficiency of a massively-parallel (GPU-like) architecture. Such architectures inherently offer both high performance and high efficiency and are well-suited to power- and energy-constrained systems, or essentially, nearly all future systems. An important issue that impacts these future designs is the problem of increasing process variation. Process variation is conservatively addressed today using wide design margins, which significantly degrade energy-efficiency, limiting performance and power scaling. We propose, discuss, and evaluate a collection of cooperative microarchitectural and circuit level mechanisms to trim these margins effectively, without sacrificing neither performance nor correctness. Our mechanisms include techniques that, for the first time, enable effective timing speculation in a parallel SIMD pipeline and that address the specific concerns of designing for, and operating in a high-variation environment.

We first show that process variation poses unique problems in the context of a parallel SIMD pipeline. We propose the decoupled parallel SIMD pipeline approach (DPSP) combined with pipeline weaving to handle input-dependent variation and and static process variation impact, respectively. To

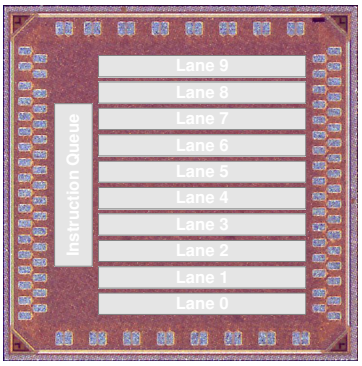


Figure 6.1: Synctium test-chip [37, 58] die photo.

validate the circuit components of these approaches we collaborated on designing and fabricating a test-chip (Figure 6.1). This test-chip was fabricated in a $45nm$ IBM SOI process by our collaborators [58] and can operate at reduced voltage to emphasize the impact of variation. Our collaborators are currently evaluating the applicability of our architectural ideas in the silicon prototype.

We show that prior timing-speculation mechanisms, which were designed for sequential pipelines, are insufficient to reduce design margins speculatively in a SIMD pipeline. When multiple pipelines lanes operate in parallel and in lockstep, the performance and efficiency implications of error events are much more significant than in a sequential pipeline. We address this with the decoupled parallel SIMD pipeline (DPSP), which enables effective timing speculation in a parallel pipeline.

We evaluated the architectural implications of DPSP within the context of a GPU. This initial evaluation was limited in two ways that we would like to extend in future work. First, we used a Razor-like error detection

mechanism with recovery by stall. We used the stall technique because it is made possible by DPSP partitioning the SIMD design into lanes that are small and because stall-based recovery has a smaller impact on performance than the alternatives. As future research we suggest evaluating the gain due to DPSP with a recovery implemented by flush and replay. Avoiding recovery by stall will enable utilizing simpler variants of a Razor latch, such as Razor II [11], TDTB [4], or DSTB [4].

The second extension we plan is to evaluate a GPU architecture that supports both local scratchpad memory and a first-level cache. Our work so far used a GPU model with only a local scratchpad, and thus error events and decoupling impact the memory access characteristics of the GPU. With a first-level cache, DPSP will not impact the off-chip memory accesses because the cache provides much greater flexibility in memory coalescing. DPSP may, however, change the order in which cache accesses are performed. This reordering may have detrimental impact on performance because of a greater potential for cache bank conflicts and because it is possible that additional cache accesses will be necessary. These detrimental effects can be mitigated either by synchronizing before each memory operation similarly to the technique shown in Section 4.5 for memory coalescing or by adding microarchitectural mechanisms that provide buffering that can be used to eliminate the problems caused by reordering.

With respect to defects introduced in fabrication that may slow down all the lanes in the SIMD pipeline, we propose a new form of sparing we term

pipeline weaving. With no sparing, the slowest lane determines the operating point of the entire SIMD pipeline. With traditional sparing at lane granularity, it is still an entire lane that determines the operating parameters. With pipeline weaving, each pipeline stage is spared independently, enabling much finer control over choosing the optimal operating point. The use of the weave interconnect topology keeps the overheads of this flexible approach very low. These overheads can be measured and analyzed using the test-chip. We did not conduct these measurements in the context of this dissertation because of issues relating to the yield of the fabricated parts.

Finally, we develop a new error detection mechanism (DDE) that, unlike Razor, allows speculative design-margin reduction in the presence of severe process variation. Using two designs that are diversified such that a timing violation does not simultaneously occur in the same output of the two circuits simultaneously, we can detect all errors, even when variation is extreme. Thus, the efficiency of such a mechanism depends on the amount of diversity between the two designs. We provide metrics to evaluate the diversity and list different levels at which diversity could be achieved. Future research is required to develop systematic approaches to obtain such diversity. Furthermore, we suggest as additional potential future research to evaluate the overheads due to stall recovery as well as its comparison to flush and replay in the context of DDE.

The DDE approach, as presented, however, is not limited to a SIMD pipeline and can be used in high-reliability designs that utilize modular re-

dundancy. Using diversity will improve the efficiency of such systems through by speculatively decreasing the supply voltage or increasing frequency. Modular redundancy already provides the components necessary for detection and recovery, and the overhead of DDE is thus minimal, while the gains are potentially significant (as discussed in Chapter 5).

Due to high energy efficiency of the SIMD-based pipeline, products across the board, from cell phones to super computers are utilizing this execution style. In the context of small form factor devices, applied techniques discussed will allow improving battery life because energy consumption is reduced. Further, some mobile, embedded, and even, implantable electronics face strict power dissipation constraints in addition to energy consumption constraints. Such systems will also benefit from our techniques, which enable operation near or below the threshold voltage of the transistors, and require overhead only when such operation is truly required. In the context of high-end supercomputers, using these approaches will allow improving performance-per-watt that can be used to increase the overall performance of these power-constrained systems and/or reduce their costly energy consumption.

DPSP, weaving, and DDR, however, introduce some software-related challenges. Using the proposed techniques will increase the dependency of the performance and energy consumption of an application in the input data. Moreover, unlike today, performance obtained while processing the same input

data on different units may differ. As a result it may require changes in the scheduling and load balancing methods used to avoid bottlenecks.

Appendices

Appendix A

Test Chip Specifications

In this appendix, we present the specifications for the test chip designed [58] to evaluate the ideas of *decoupled parallel SIMD pipeline* (DPSP) and *pipeline weaving* presented in Chapter 3.

A.1 Test-Chip Overview

This test-chip, is our first attempt to implement and to test on a real silicon the behavior of the architectural-circuit techniques we propose. Table A.1 summarizes the properties of the test-chip. Unfortunately due to limited time and resources, we would have to limit the scope of the techniques we will test to (sorted by ABC):

- Decoupling interface
- Failure detection using Razor-like technique in near-threshold environment
- Frequency/ energy (power)/ V_{dd}/ performance interactions
- Proof of concept

- Variability as function of V_{dd}/ Frequency
- Weaving

Table A.1: Properties of the test-chip

Property	Value
Technology	$45nm$
Area	$2mm$ x$2mm$
V_{dd}	0.3V-1V
Frequency	100kHz - 100MHz
pins	40-50
Data width	16-bit
Lanes	10

In addition to the test-chip, we are going to build a complementary test-board that would be used to interface between the test-chip and the host computer. In addition it might contain some BIST mechanisms on board. However, the scope of this document is limited to specifications of the test-chip only.

A.2 Test-Chip Architecture

While the design of Synctium involves 16 *functional* lanes and a sequencer, the design of the test-chip will contain only 8 functional lanes. To allow weaving in ratio of 1:4, i.e., 1 extra functional unit for each 4, it will summarize to 10 total lanes.

Each lane is organized as shown in Figure A.1. It contains 3 pipe-stages separated in between by Razor flip-flops (will be referred to as razor from this

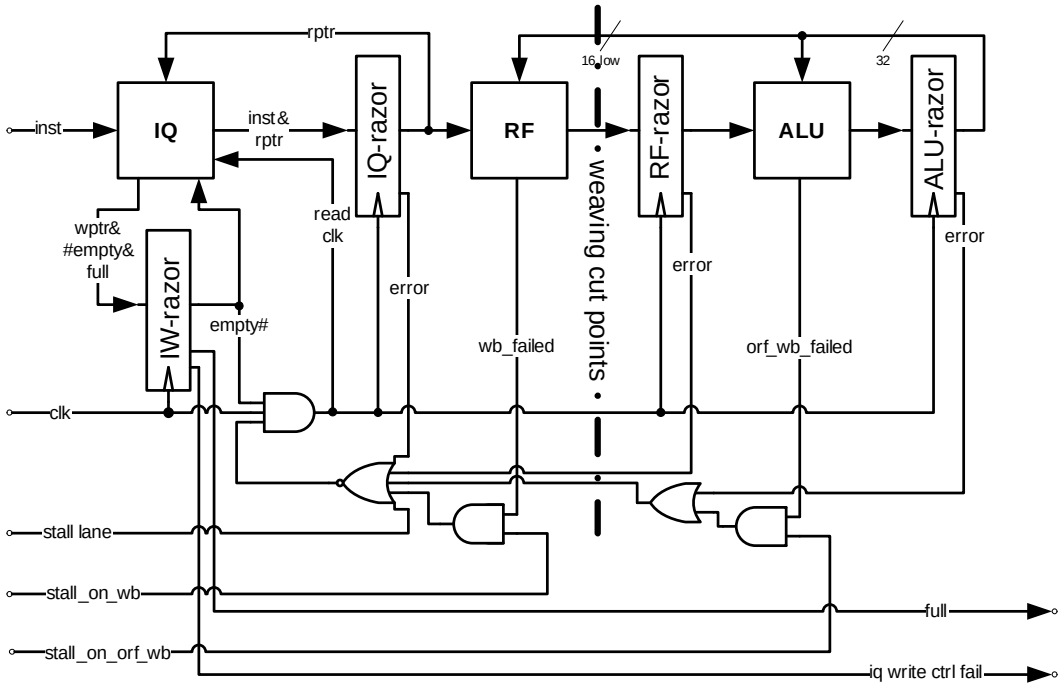


Figure A.1: The test-chip single lane micro architecture, note that not all the signals are shown on this diagram.

point on). Each razor (see Section 3.3.1.2) is responsible for detecting failures of the previous pipe-stage. In case of failure it signals 'error' which postpones the next cycles for all the razor flip-flops besides the special razor coupled to IQ.

The IQ stage implements an instruction queue (see Section A.3). It gets the next instruction to be fetched as an input for the tester and can output a 'full' signal. While the rest of the razors are gated by $\Sigma error$ indication and the 'empty' status of the IQ, there is a need to allow 'fetch into IQ' operations even while the above occurs. The non-gated razor coupled with IQ pipe-stage

serves for that purpose. Another way to stall the back-end of the lane while allowing the IQ to fetch-in new instructions is by setting the 'stall_lane' control signal that would gate the clock for the back-end pipe-stages.

The RF stage is the only stage in which each instruction passes twice (see Section A.4). Once to read the RF (register file) and once to WB (write-back the result). While a failure of RF read would be detected by the following razor, the failure on WB should be detected by other methods.

The ALU (or EX stage) is basically the ALU that executes the arithmetical operations (see Section A.5). While most of the operations have 2 sources, there is a special case of MADD (multiply add) that uses 3 sources in the following manner : $dst = src0 * src1 + src2$. Note that while $src0$ and $src1$ are 16-bit wide, $src2$ and dst are 32-bit wide. To support this and other operations we use the ORF (operand register file). It acts as a local 32-bit wide storage in ALU stage which requires the results also to be sent back to ALU. Note that the whole 32 bits of the result are sent back to ALU while only 16 low bits are sent to the RF.

In order to be able to isolate the pipe-stage under test we would like to have a separate V_{dd} plane for each of the pipe-stage (across all the lanes). Same goes to all the razor flip-flops across the whole chip which under some circumstances we would like to ensure their fault free operation.

The dashed line in Figure A.1 shows the cut points of weaving. As described in Section A.6 we will use a dedicated circuit to allow eliminating faulty units (ALU or IQ+RF).

Each instruction is 24-bit wide and Table A.2 summarizes the breakdown of the fields.

Table A.2: Instruction encoding

Instruction field	bit width
Opcode	4
src0	5
src1	5
dst	5
Immediate	5
Total	24

A.3 IQ Pipe-stage

In Figure A.2 we summarize the structure of the IQ pipe-stage. It contains 5 24-bit buffers (in the center) controled with read and write pointers selecting the entries for read/write. To distinguish between full and empty queue cases, we use 'color' property of each pointer. Each time a pointer wraps around the queue it flips its color bit. Thus Table A.3 shows the calculation of the '#empty' ($\overline{empty}$) and 'full' indications.

Unlike any other pipe-stage, some of the IQ inputs (and outputs) are connected through a gated flop, while others are connected through a non-gated flop. As a result, it might happen that during a 'write' cycle, there will

Table A.3: IQ #empty/full calculation

pointers match	color match	#empty	full
0	0	1	0
0	1	1	0
1	0	1	1
1	1	0	0

be no 'read' operation since the IQ-razor will be gated (either due to an error in one of the stages including IQ-read or due to the IQ being empty).

On the other hand, the case of a 'read' operation with no 'write' operation at the same time is possible only in case the provided instruction (input) will not be valid. This might happen if one of the lanes is full and the sequencer/host avoids sending in any new instructions. In this case it will be signaled using the 'instr_in_valid' indication.

For testing proposes we allow a special operation mode of the IQ as a cyclic buffer. It will be operated by setting the 'cyclic' control input to the IQ. In this mode we assume that all 5 instructions stored in the IQ are valid and these will be supplied to the lane in a continuous loop.

In Table A.4 we summarize all the IQ pipe-stage interface signals along with their origins/destinations.

There are 3 possible indications for IQ stage failure:

- iq_write_failed — indicates a failure detected on write into the queue (buffers).

Table A.4: IQ interface summary

Input		Output		
Signal	Source	Signal	Dest	Next Dest
prev full	IW-razor	write_on_full	IW-razor	DFT
prev #empty	IW-razor	instr_out_valid	IQ-razor	RF
curr wcolor	IW-razor	new wcolor	IW-razor	IQ
curr wptr	IW-razor	new wptr	IW-razor	IQ
instr_in_valid	PI	instr_out	IQ-razor	RF
instr_in	PI	iq_write_failed	IW-razor	DFT
cyclic	PI	new rptr	IQ-razor	IQ
curr rptr	IQ-razor	new ecolor	IQ-razor	IQ
curr rcolor	IQ-razor	#empty	IW-razor	IQ/logic
read clk	(gated) clk	full	IW-razor	IQ/POUT

- iq_write_ctrl_failed — indicates a failure detected by the IW-razor.

- iq_razor_error — indicates a failure detected by the IQ-razor.

A.4 RF(/WB) Pipe-stage

The RF structure is accessed twice. Once during the RF pipe-stage and once on WB. Therefore it has inputs both from IQ-razor and ALU-razor. In Table A.5 we summarize the interface signals of this structure along with their origins/destinations.

Signals 'op', 'instr_valid', and 'dst idx' are just propagated to the next stage (ALU). Signals 'src_idx' are being used in this stage and propagated to the next stage (ALU).

Table A.5: RF interface summary

Input		Output		
Signal	Source	Signal	Dest	Next Dest
instr_valid	IQ-razor	instr_valid	RF-razor	ALU
immediate	IQ-razor	immediate	RF-razor	ALU
op	IQ-razor	op	RF-razor	ALU
dst idx	IQ-razor	dst idx	RF-razor	ALU
src0 idx	IQ-razor	src0 idx	RF-razor	ALU
src1 idx	IQ-razor	src1 idx	RF-razor	ALU
wb data	ALU-razor	src0 data	RF-razor	ALU
wb valid	ALU-razor	src1 data	RF-razor	ALU
wb idx	ALU-razor	wb failed	logic	DFT

Register numbers are encoded in the 4 least significant bits of '*_idx' signals if the MSB bit is set. The MSB bit is used to indicate 'special registers' (see Table A.7). For the test-chip we do not implement a power-saving feature that would save an access to the RF in case it is not required (usage of a 'special register').

There are 2 possible indications for IQ stage failure due to short cycle time:

- wb_failed — indicates a failure detected on write into the RF.
- rf_razor_error — indicates a failure detected by the RF-razor (read).

Since this is one of the things we would like to test with this test-chip, control signal 'stall_on_wb' will determine whether 'wb_failed' indication

should be taken into account for clock gating. Moreover, this signal will also be counted (see Section A.8).

We assume at least 2 cycles between back-to-back operations (no bypass).

A.5 ALU or EX Pipe-stage

The main function of the ALU pipe-stage is to compute the arithmetic operation specified by the instruction. While most of the operations consume 2 sources (src0, src1), MADD operation also consumes a third source — src2. Widths of src0 and src1 are 16-bit, while the width of the result is 32-bit in case of MUL flavor instructions or 16-bit zero extended to 32-bit otherwise. Since src2 is used always in conjunction with MUL, its with is also 32-bit.

Since the data-width of the lane is 16-bit, we use the ORF structure to supply the data for src2. The ORF is built out of two 16-bit wide buffers also referred to as ORF_LO and ORF_HI. The 32-bit result is fed back into the ALU to be used if the destination is ORF. Only the lower 16bit of the result are propagated to the WB stage (RF) to be written into actual registers.

Similarly to previous stages, ALU also has two signals that it just propagates as-is : 'instr(or wb)valid' and 'dst idx'. All signals generated by ALU (besides 'orf_wb_failed') are routed to the ALU-razor and from there back to ALU and RF. All inputs of ALU are coming either from RF-razor or ALU-razor. In Table A.6 we summarize all the ALU interface signals.

Table A.6: ALU interface summary

Input		Output		
Signal	Source	Signal	Dest	Next Dest
insrt(wb)_valid	RF-razor	insrt(wb)_valid	ALU-razor	ALU/RF
dst idx	RF-razor	dst idx	ALU-razor	ALU/RF
src0 idx	RF-razor	result	ALU-razor	ALU/RF
src0 data	RF-razor	orf_wb_failed	logic	DFT
immediate	RF-razor			
src1 idx	RF-razor			
src1 data	RF-razor			
wb idx	ALU-razor			
wb valid	ALU-razor			
wb data	ALU-razor			
op	RF-razor			

The muxes shown in Figure A.4 select the data out of the different possible options according to Table A.7. Since the width of the immediate is only 5 bits and the width of src0/1 is 16-bit we use zero extension (11-bit MSB=0, 5-bit LSB=immediate). The destination is encoded the same way although it can not have the value of '01000' which stands for immediate (see Table A.7).

Table A.7: Source/Destination Encoding

Source/Destination Index	Meaning
1abcd	register 'abcd'
01000	immediate
00100	ORF
00010	ORF_LO
00001	ORF_HI
00000	reserved

There are 2 possible indications for ALU stage failure due to short cycle time:

- orf_wb_failed — indicates a failure detected on write into the ORF.
- alu_razor_error — indicates a failure detected by the ALU-razor.

A.6 Weaving

One of the main features of the test-chip is evaluating weaving. As explained in Section 3.4.2, the idea behind weaving is to cross-connect adjacent functional units to allow fine grain sparing.

To avoid excessive use of wires, we are using the same wires between the adjacent units for both directions as shown in Figure A.5.

The basic weaving unit ("brick") is presented in Figure A.6. It routes data from left (unit A) to right (unit B) and depending on the configuration, might also route the data to the unit above or below through up/down interfaces (as shown in Figure A.5). The destination to which the data is routed

Table A.8: Instructions Opcodes Encoding

Op.	Instr.	Explanation	Width	Comment
0000	ADD	$result := src0 + src1$	16-bit	
0001	SUB	$result := src0 - src1$	16-bit	
0010	MUL	$result := src0 * src1$	16-bit	Signed multiply
0011	UMUL	$result := src0 * src1$	16-bit	Unsigned multiply
0100	MADD	$result := src2 + src0 * src1$	32-bit	d = a+b*c, signed multiply
0101	MSUB	$result := src2 - src0 * src1$	32-bit	d = a-b*c, singed multiply
0110	UMADD	$result := src2 + src0 * src1$	32-bit	d = a+b*c, unsigned multiply
0111	UMSUB	$result := src2 - src0 * src1$	32-bit	d = a-b*c, unsigned multiply
1000	SHIFT	$result := src0 \ll src1$	16-bit	Logical shift left (no sign extension)
1001	ASHIFT	$result := src0 \ll src1$	16-bit	Arithmetic shift left (sign extension)
1010	ROT	$result := src0 \ll src1$	16-bit	Rotate left
1011	AND	$result := src0 \& src1$	16-bit	
1100	OR	$result := src0 \| src1$	16-bit	
1101	XOR	$result := src0 \oplus src1$	16-bit	
1110	NOT	$result := src0$	16-bit	Single source instruction
1111	TEQ	$result :=$ ($src0 == src1$? 0xFF:0x00)	16-bit	Compare to a value and set mask to true if equal

by the brick is controlled by control signals wctl0–3 according to Table A.9. The rest of the configuration bits combinations are forbidden.

Weaving brick as shown in Figure A.6 provides only routing from left to right i.e. from unit A to unit B. However, in our case there is also a data feedback from unit B (ALU) to unit A (RF). Figure A.7 shows the usage of the weaving brick for bidirectional communication between unit A and unit B. Relaying on symmetry properties, the backward routing brick(from unit B

Table A.9: Allowed weaving brick routing configurations (disc. stands for disconnected)

Routing		Control Bits			
unit A to:	unit B from:	wctrl0	wctrl1	wctrl2	wctrl3
B	A	0	0	0	0
down	disc.	0	0	0	1
disc.	down	0	0	1	0
disc.	up	0	1	0	0
down	up	0	1	0	1
up	disc.	1	0	0	0
up	down	1	0	1	0

to unit A) can be controlled using the same signals that control the forward routing brick as shown in Figure A.7.

Moreover, vertically adjacent bricks can share 2 control signals. Thus, weaving of n lanes can be controlled using $2(n-1)$ signals as shown in Figure A.8. Unfortunately, due to size limitations, Figure A.8 shows weaving of 4 lanes only while in test-chip we will use the same method for weaving of each 5 lanes.

All the signals cut through by the dashed line in Figure A.1 should be weaved. Table A.10 summarizes all the signals to be weaved.

A.7 Reset

Reset signal should be routed to all the filp-flops (including razors). All the control signals (located in IQ pipe-stage – see Section A.3) like the readb/

Table A.10: Weaved signals

		Direction	
Signal	Width[bit]	RF→ALU	ALU→RF
instr_valid	1	X	
immediate	5	X	
op	4	X	
dst idx	5	X	
src0 idx	5	X	
src1 idx	5	X	
src0 data	16	X	
src1 data	16	X	
gated_clk	1	X	
rf_razor_error	1		X
alu_error	1		X
wb data	16		X
wb valid	1		X
wb idx	5		X

write pointers and empty indication are planned to reset on zero values. For instance upon reset, read/write pointers would point to the first IQ entry and empty indication will be cleared. The back-end (RF/ALU) upon reset will have instr/wb_valid indication cleared that would disable any write-back.

A.8 DFT

To allow easier testing of the system we define a set of control signals and counters. Unlike the scan-chains to access the PI/POUT and intermediate flops (razors), control signals will be changed much less frequently. Usually

these would be set once in the beginning of the experiment. Counters might be accessed during the experiment or at its end to check occurrence of different events during the period. All the counters are 8-bit counters (reset to 0 upon global reset). While all the counters and most of the control signals are per lane, there are some control signals (like reset and weaving control) which are unique. All the counters and control signals which are replicated for each lane, are followed by the index of the lane (like cyclic_3 that stands for lane3 IQ operating in cyclic buffer mode). Table A.12 summarizes the counters and Table A.11 summarizes the control signals.

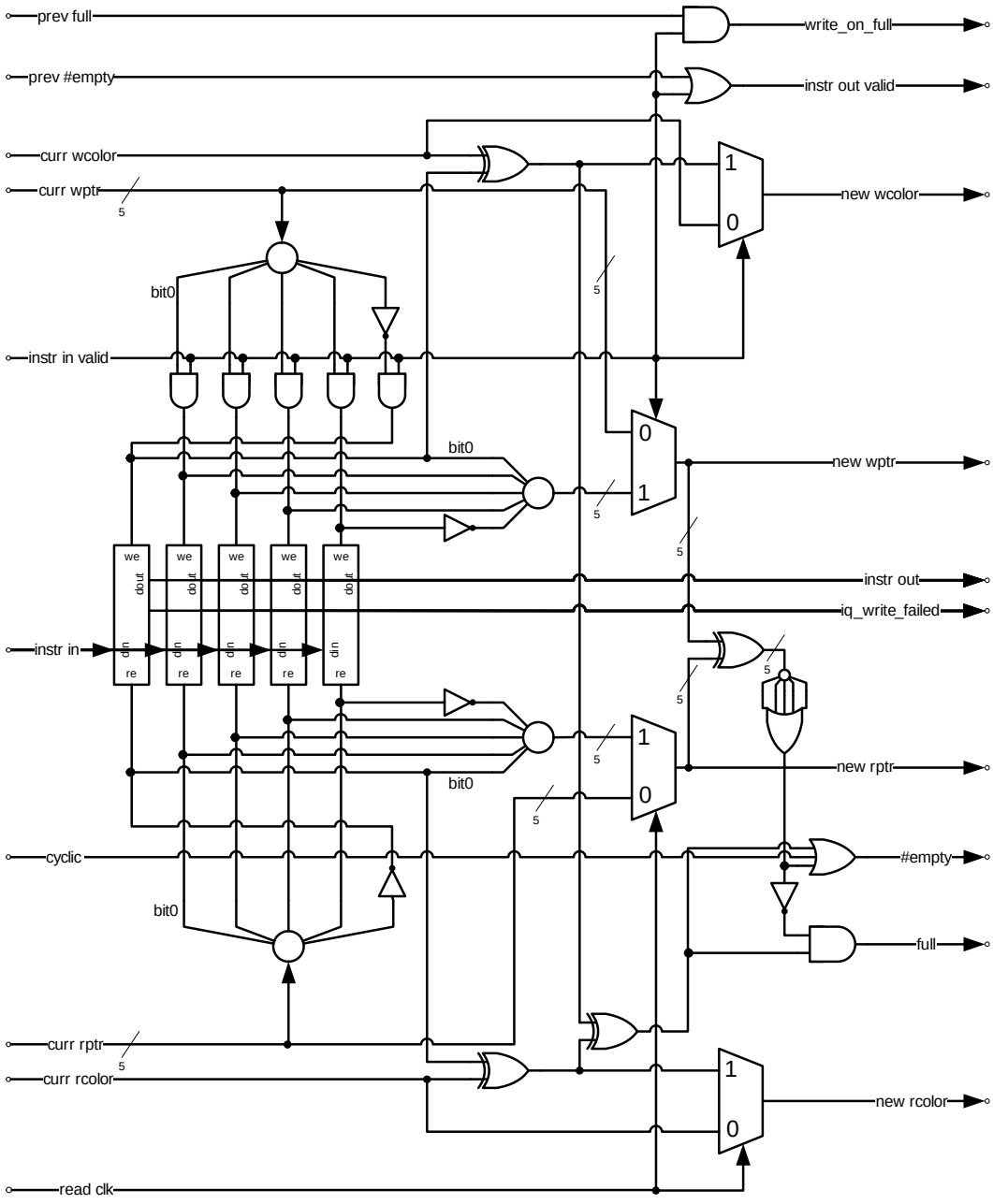


Figure A.2: IQ Pipe-stage structure

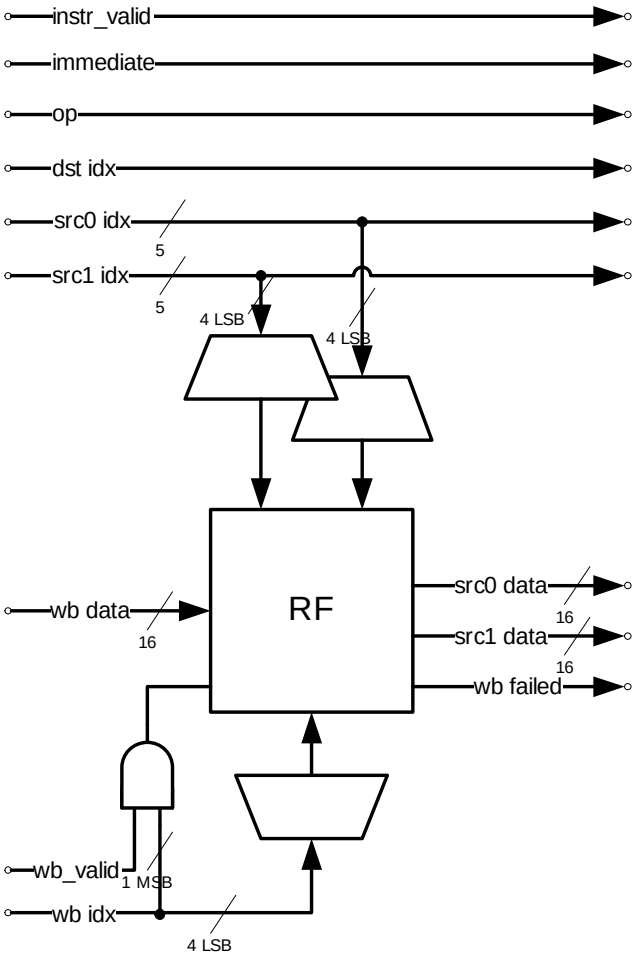


Figure A.3: RF(/WB) Pipe-stage structure

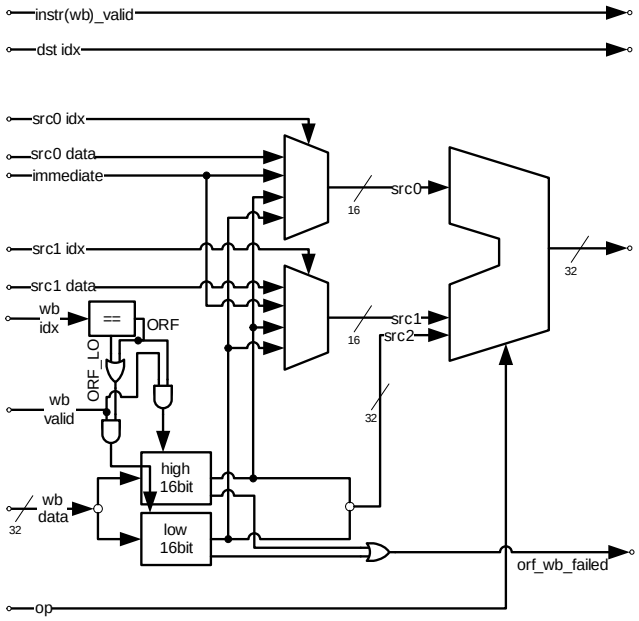
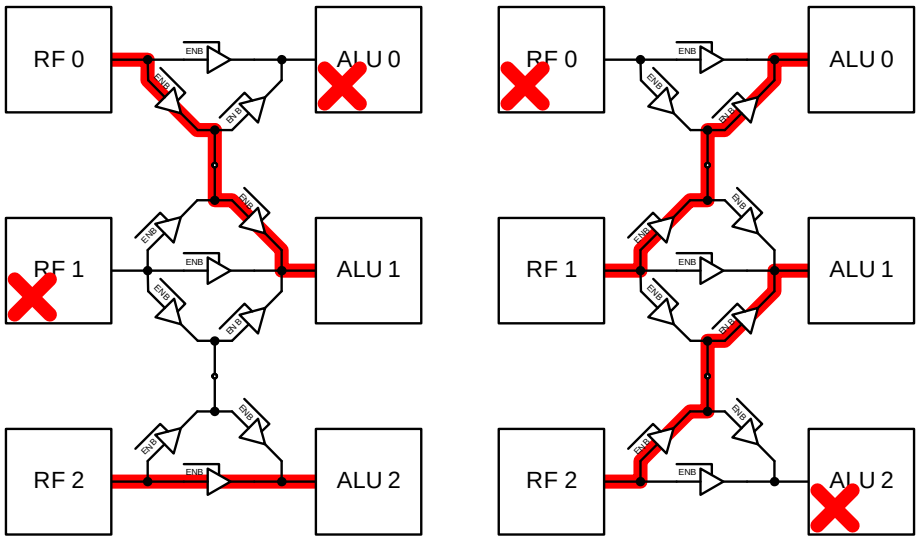


Figure A.4: ALU(/EX) Pipe-stage structure



(a) Example 1 (b) Example 2

Figure A.5: Weaving examples showing bidirectional usage of the same wire in different configurations.

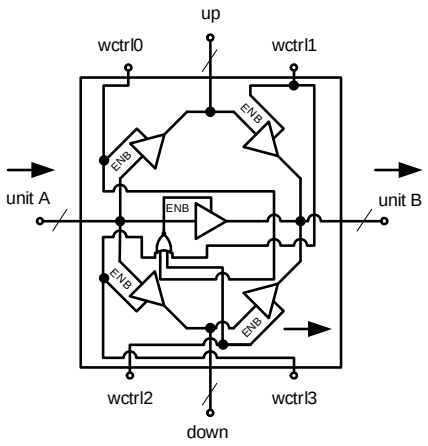


Figure A.6: Basic weaving "brick". Note that the direct propagating tristate is controlled by NOR of all other 4 controls, i.e. operates iff all the rest tristates are disabled.

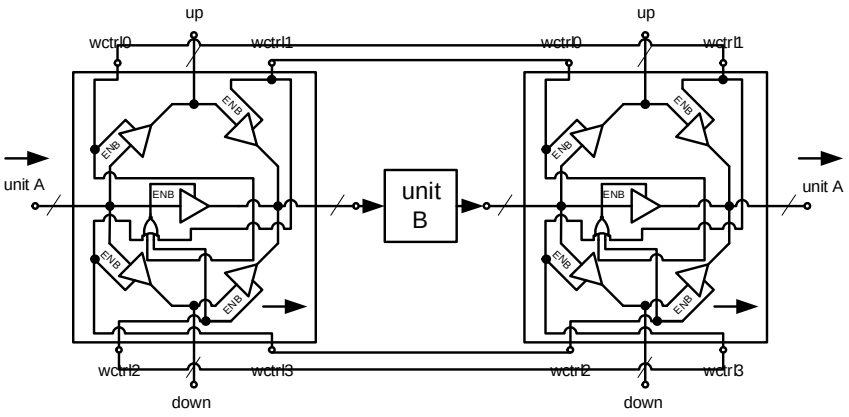


Figure A.7: Bidirectional weaving.

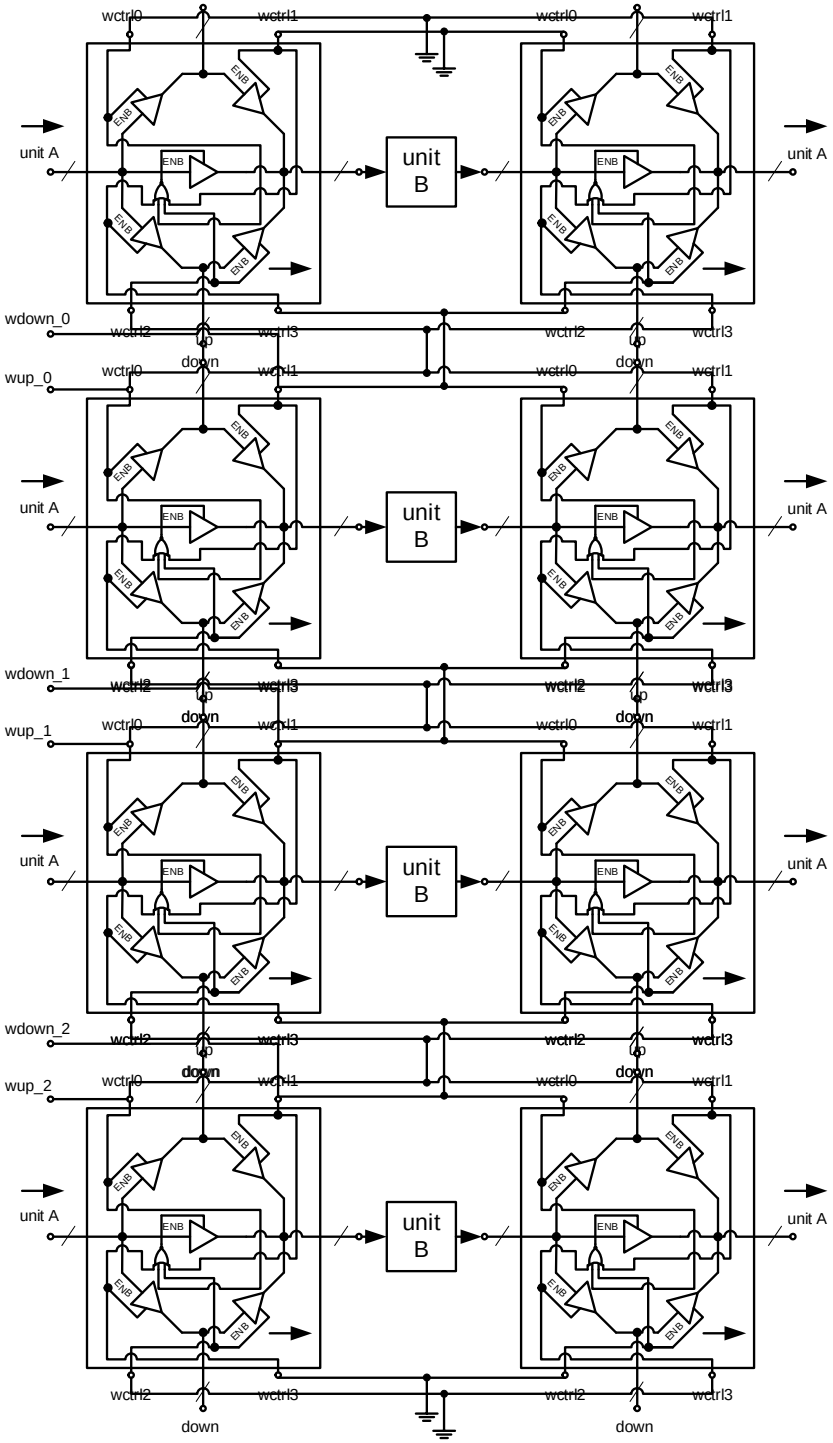


Figure A.8: Weaving 4 lanes

Table A.11: Control signals

Control signal name	Granularity	Meaning
cyclic	lane	IQ operates as cyclic buffer
stall_lane	lane	Back-end clock is gated (can be used to fill in IQ)
stall_on_wb	lane	Gate BE clock on RF wb failure
stall_on_orf_wb	lane	Gate BE clock on ORF wb failure
disable_IQRF	lane	Disable IQ and RF (until the weaving cut point) - can be used with weaving
disable_ALU	lane	Disable ALU (after the weaving cut point) - can be used with weaving
wup	lane*	Weaving control (only for lanes 0–3 and 5–8)
wdown	lane*	Weaving control (only for lanes 0–3 and 5–8)
ls_FE_A	chip	Lock-stepped instruction fetch for lanes 0–4 (lanes 1–4 PIs are connected to lane 0 PIs)
ls_FE_B	chip	Lock-stepped instruction fetch for lanes 0–4 (lanes 1–4 PIs are connected to lane 0 PIs)
ls_FE_AB	chip	Lock-stepped instruction fetch for lanes 0 and 5 (lane5 PIs are connected to lane 0 Pis - in conjunction with other ls_FE signals and make the whole chip to fetch in lockstep mode)
reset	chip	Reset the chip

Table A.12: DFT counters

Counted Signal	Meaning	Detected at
iq_write_ctrl_failed	Failure in instruction write to IQ	IQ buffer
write_on_full	Instruction fetched which IQ is full	IQ logic
iq_write_failed	Failure in instruction write to IQ	IW-razor
iq_razor_error	Failure in instruction read from IQ	IQ-razor
wb_failed	Failure in write-back	RF buffer
rf_razor_error	Failure in RF read	RF-razor
orf_wb_failed	Failure in ORF write-back	ALU logic
alu_razor_error	Failure in ALU logic	ALU-razor

Bibliography

[1] A. Bakhoda, G.L. Yuan, W.W.L. Fung, H. Wong, and T.M. Aamodt. Analyzing CUDA workloads using a detailed GPU simulator. In *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pages 163–174. IEEE, 2009.

[2] K. Bernstein, DJ Frank, AE Gattiker, W. Haensch, BL Ji, SR Nassif, EJ Nowak, DJ Pearson, and NJ Rohrer. High-performance cmos variability in the 65-nm regime and beyond. *IBM journal of research and development*, 50(4.5):433–449, 2006.

[3] WJ Bouknight, S.A. Denenberg, D.E. McIntyre, JM Randall, A.H. Sameh, and D.L. Slotnick. The illiac iv system. *Proceedings of the IEEE*, 60(4):369–388, 1972.

[4] K. Bowman, J. Tschanz, N. Kim, J. Lee, C. Wilkerson, S.L. Lu, T. Karnik, and V. De. Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance. *IEEE J. Solid-State Circuits*, 44(1): 49–63, 2009.

[5] K. Bowman, J. Tschanz, C. Wilkerson, S.L. Lu, T. Karnik, V. De, and S. Borkar. Circuit techniques for dynamic variation tolerance. In *Pro-*

ceedings of the 46th Annual Design Automation Conference, pages 4–7. ACM, 2009.

[6] D. Bull, S. Das, K. Shivashankar, G.S. Dasika, K. Flautner, and D. Blaauw. A power-efficient 32 bit arm processor using timing-error detection and correction for transient-error tolerance and adaptation to pvt variation. *Solid-State Circuits, IEEE Journal of*, 46(1):18–31, 2011.

[7] B.H. Calhoun and A.P. Chandrakasan. Ultra-dynamic voltage scaling (udvs) using sub-threshold operation and local voltage dithering. *Solid-State Circuits, IEEE Journal of*, 41(1):238–245, 2006.

[8] Y. Cao, P. Gupta, AB Kahng, D. Sylvester, and J. Yang. Design sensitivities to variability: extrapolations and assessments in nanometer vlsi. In *ASIC/SOC Conference, 2002. 15th Annual IEEE International*, pages 411–415. IEEE, 2002.

[9] A.P. Chandrakasan, M. Potkonjak, J. Rabaey, and R.W. Brodersen. Hyper-lp: A system for power minimization using architectural transformations. In *Computer-Aided Design, 1992. ICCAD-92. Digest of Technical Papers., 1992 IEEE/ACM International Conference on*, pages 300–303. IEEE, 1992.

[10] J. Crop, E. Krimer, N. Moezzi-Madani, R. Pawlowski, T. Ruggeri, P. Chiang, and M. Erez. Error Detection and Recovery Techniques for Variation-Aware CMOS Computing: A Comprehensive Review. *Journal of Low Power Electronics and Applications*, 1(3):334–356, 2011.

[11] S. Das, C. Tokunaga, S. Pant, W.H. Ma, S. Kalaiselvan, K. Lai, D. Bull, and D. Blaauw. RazorII: In situ error detection and correction for PVT and SER tolerance. *Solid-State Circuits, IEEE Journal of*, 44(1):32–48, 2009.

[12] M.E. Dean. *STRiP: a self-timed RISC processor*. PhD thesis, Stanford University, 1992.

[13] M.E. Dean, T.E. Williams, and D.L. Dill. Efficient self-timing with level-encoded 2-phase dual-rail (ledr). In *Proceedings of the 1991 University of California/Santa Cruz conference on Advanced research in VLSI*, pages 55–70. MIT Press, 1991.

[14] M.E. Dean, D.L. Dill, and M. Horowitz. Self-timed logic using current-sensing completion detection (cscd). *The Journal of VLSI Signal Processing*, 7(1):7–.16, 1994.

[15] A. Drake, R. Senger, H. Deogun, G. Carpenter, S. Ghiasi, T. Nguyen, N. James, M. Floyd, and V. Pokala. A distributed critical-path timing monitor for a 65nm high-performance microprocessor. In *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 398–399. IEEE, 2007.

[16] D. Ernst, S. Das, S. Lee, D. Blaauw, T. Austin, T. Mudge, N.S. Kim, and K. Flautner. Razor: circuit-level correction of timing errors for low-power operation. *IEEE MICRO*, pages 10–20, 2004.

[17] J.A. Fisher. Very Long Instruction Word architectures and the ELI-512. In *Proceedings of the 10th annual international symposium on Computer architecture*, ISCA '83, pages 140–150, 1983.

[18] M.J. Flynn. Very high-speed computing systems. *Proceedings of the IEEE*, 54(12):1901–1909, 1966.

[19] P. Franco and E.J. McCluskey. On-line delay testing of digital circuits. In *VLSI Test Symposium, 1994. Proceedings., 12th IEEE*, pages 167–173, apr 1994.

[20] P. Friedberg, Y. Cao, J. Cain, R. Wang, J. Rabaey, and C. Spanos. Modeling within-die spatial correlation effects for process-design co-optimization. In *Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on*, pages 516–521. IEEE, 2005.

[21] W.W.L. Fung and T.M. Aamodt. Thread Block Compaction for Efficient SIMT Control Flow. In *17th International Symposium on High Performance Computer Architecture (HPCA-17)*, February 2011.

[22] S.B. Furber, J.D. Garside, and D.A. Gilbert. Amulet3: A high-performance self-timed arm microprocessor. In *Computer Design: VLSI in Computers and Processors, 1998. ICCD'98. Proceedings. International Conference on*, pages 247–252. IEEE, 1998.

[23] S. Gochman, A. Mendelson, A. Naveh, and E. Rotem. Introduction to intel core duo processor architecture. *Intel Technology Journal*, 10(2): 89–97, 2006.

[24] S. Gupta, S. Feng, A. Ansari, J. Blome, and S. Mahlke. The StageNet fabric for constructing resilient multicore systems. In *Proceedings of the 2008 41st IEEE/ACM International Symposium on Microarchitecture*, pages 141–151. IEEE Computer Society, 2008.

[25] S. Gupta, A. Ansari, S. Feng, and S. Mahlke. StagWeb: Interweaving pipeline stages into a wearout and variation tolerant cmp fabric. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 101–110. IEEE, 2010.

[26] T.R. Halfhill. Transmeta breaks x86 low-power barrier vliw chips use hardware-assisted x86 emulation. *Microprocessor Report*, 14(2), 2000.

[27] S. Hauck. Asynchronous design methodologies: An overview. *Proceedings of the IEEE*, 83(1):69–93, 1995.

[28] K. He, A. Gerstlauer, and M. Orshansky. Controlled timing-error acceptance for low energy idct design. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011*, pages 1–6. IEEE, 2011.

[29] S. Hong and H. Kim. An integrated gpu power and performance model. In *Proceedings of the 37th annual international symposium on Computer architecture ISCA 10*. ACM Press, 2010.

[30] ITRS. Process Integration, Devices, and Structures (PIDS) Update, 2010. URL `http://www.itrs.net/links/2010itrs/home2010.htm`.

[31] A.B. Kahng, B. Li, L.S. Peh, and K. Samadi. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 423–428, 2009.

[32] G. Karakonstantis, D. Mohapatra, and K. Roy. System level dsp synthesis using voltage overscaling, unequal error protection & adaptive quality tuning. In *Signal Processing Systems, 2009. SiPS 2009. IEEE Workshop on*, pages 133–138. IEEE, 2009.

[33] H. Kaul, M.A. Anders, S.K. Mathew, S.K. Hsu, A. Agarwal, R.K. Krishnamurthy, and S. Borkar. A 300mV 494GOPS/W Reconfigurable Dual-Supply 4-Way SIMD Vector Processing Accelerator in 45nm CMOS. In *IEEE International Solid-State Circuits Conference, 2009. ISSCC 2009. Digest of Technical Papers*, pages 260–261, 2009.

[34] R. Kay and L. Pileggi. Primo: probability interpretation of moments for delay calculation. In *Proceedings of the 35th annual Design Automation Conference*, pages 463–468. ACM, 1998.

[35] A. Kondratyev and K. Lwin. Design of asynchronous circuits using synchronous cad tools. *Design & Test of Computers, IEEE*, 19(4):107–117, 2002.

[36] I. Koren and AD Singh. Fault tolerance in VLSI circuits. *Computer*, 23 (7):73–83, 1990.

[37] E. Krimer, R. Pawlowski, M. Erez, and P. Chiang. Synctium: a near-threshold stream processor for energy-constrained parallel applications. *IEEE IEEE Computer Architecture Letters*, pages 21–24, 2010.

[38] E. Krimer, P. Chiang, and M. Erez. Lane decoupling for improving the timing-error resiliency of wide-simd architectures. In *to appear in the International Symposium on Computer Architecture (ISCA12)*, June 2012.

[39] V.R. Lesser, J. Pavlin, and E. Durfee. Approximate processing in real-time problem solving. *AI magazine*, 9(1):49, 1988.

[40] S. Li, J.H. Ahn, R.D. Strong, J.B. Brockman, D.M. Tullsen, and N.P. Jouppi. McPAT: an integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Microarchitecture, 2009. MICRO-42. 42nd Annual IEEE/ACM International Symposium on*, pages 469–480. IEEE, 2009.

[41] X. Liang, G.Y. Wei, and D. Brooks. Revival: A variation-tolerant architecture using voltage interpolation and variable latency. In *Proceedings of the 35th International Symposium on Computer Architecture*, pages 191–202. IEEE Computer Society, 2008.

[42] K.J. Lin, S. Natarajan, J.W.S. Liu, United States. National Aeronautics, and Space Administration. Imprecise results: Utilizing partial computa-

tions in real-time systems. In *Proceedings of the Eighth Real-Time Systems Symposium (San Jose, CA)*, pages 210–217, 1987.

[43] J.T. Ludwig, S.H. Nawab, and A.P. Chandrakasan. Low-power digital filtering using approximate processing. *Solid-State Circuits, IEEE Journal of*, 31(3):395–400, 1996.

[44] A.J. Martin. Towards an energy complexity of computation. *Information Processing Letters*, 77(2-4):181–187, 2001.

[45] A.J. McAuley. Four state asynchronous architectures. *Computers, IEEE Transactions on*, 41(2):129–142, 1992.

[46] S. Mitra, N.R. Saxena, and E.J. McCluskey. A design diversity metric and reliability analysis for redundant systems. In *Test Conference, 1999. Proceedings. International*, pages 662–671. IEEE, 1999.

[47] E. Mizan, T. Amimeur, and M.F. Jacome. Self-imposed temporal redundancy: An efficient technique to enhance the reliability of pipelined functional units. In *Computer Architecture and High Performance Computing, 2007. SBAC-PAD 2007. 19th International Symposium on*, pages 45–53. IEEE, 2007.

[48] J.C. Murtha. Highly parallel information processing systems. *Advances in Computers*, 7:1–116, 1966.

[49] S. Mutoh, T. Douseki, Y. Matsuya, T. Aoki, S. Shigematsu, and J. Yamada. 1-v power supply high-speed digital circuit technology with

multithreshold-voltage cmos. *Solid-State Circuits, IEEE Journal of*, 30 (8):847–854, 1995.

[50] S.H. Nawab and E. Dorken. A framework for quality versus efficiency tradeoffs in stft analysis. *Signal Processing, IEEE Transactions on*, 43 (4):998–1001, 1995.

[51] S.H. Nawab, A.V. Oppenheim, A.P. Chandrakasan, J.M. Winograd, and J.T. Ludwig. Approximate signal processing. *The Journal of VLSI Signal Processing*, 15(1):177–200, 1997.

[52] J. Nickolls, I. Buck, M. Garland, and K. Skadron. Scalable parallel programming with CUDA. *Queue*, 6(2):40–53, 2008.

[53] K. Nose, M. Hirabayashi, H. Kawaguchi, S. Lee, and T. Sakurai. Vth-hopping scheme to reduce subthreshold leakage for low-power processors. *Solid-State Circuits, IEEE Journal of*, 37(3):413–419, 2002.

[54] NVIDIA Corp. Fermi CUDA Architecture, 2009. URL `http://www.nvidia.com/object/fermi_architecture.html`.

[55] NVIDIA Corp. Tesla C2050 and C2070 Computing Processor Board, 2010. URL `http://www.nvidia.com/docs/IO/43395/BD-04983-001_v04.pdf`.

[56] J. Park, S. Kwon, and K. Roy. Low power reconfigurable dct design based on sharing multiplication. In *Acoustics, Speech, and Signal Processing*

(ICASSP), 2002 IEEE International Conference on, volume 3, pages III–3116. IEEE, 2002.

[57] J.H. Patel and L.Y. Fung. Concurrent error detection in alu's by recomputing with shifted operands. *Computers, IEEE Transactions on*, 100(7): 589–595, 1982.

[58] R. Pawlowski, E. Krimer, J. Crop, J. Postman, N. Moezzi-Madani, M. Erez, and P. Chiang. Synctium-I: A 530mV, 10-lane SIMD Processor with Variation Resiliency in 45nm-SOI. In *Solid-State Circuits Conference, 2012. ISSCC 2012. Digest of Technical Papers. IEEE International*, feb. 2012.

[59] P. Penzes. *The design of high performance asynchronous circuits for the Caltech MiniMIPS processor*. PhD thesis, MS Thesis. Caltech, 1998.

[60] G.A. Reis, J. Chang, N. Vachharajani, R. Rangan, and D.I. August. Swift: Software implemented fault tolerance. In *Proceedings of the international symposium on Code generation and optimization*, pages 243–254. IEEE Computer Society, 2005.

[61] D.A. Reynolds and G. Metze. Fault detection capabilities of alternating logic. *Computers, IEEE Transactions on*, 100(12):1093–1098, 1978.

[62] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand. Leakage current mechanisms and leakage reduction techniques in deep-submicrometer cmos circuits. *Proceedings of the IEEE*, 91(2):305–327, 2003.

[63] T. Sakurai and A.R. Newton. Alpha-power law MOSFET model and its applications to CMOS inverter delay and other formulas. *Solid-State Circuits, IEEE Journal of*, 25(2):584–594, 1990. ISSN 0018-9200.

[64] S.R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas. VARIUS: A model of process variation and resulting timing errors for microarchitects. *Semiconductor Manufacturing, IEEE Transactions on*, 21(1):3–13, 2008.

[65] C.L. Seitz. System timing. *Introduction to VLSI systems*, pages 218–262, 1980.

[66] J.J. Shedletsky. Error correction by alternate-data retry. *Computers, IEEE Transactions on*, 100(2):106–112, 1978.

[67] N.P. Singh. A design methodology for self-time systems. 1981.

[68] K. Skaugen. Petascale to exascale: extending Intels HPC commitment, 2010. URL `http://download.intel.com/pressroom/archive/reference/ISC_2010_Skaugen_keynote.pdf`.

[69] R.F. Sproull, I.E. Sutherland, and C.E. Molnar. The counterflow pipeline processor architecture. *Design Test of Computers, IEEE*, 11(3):48, autumn/fall 1994.

[70] J.E. Stine, I. Castellanos, M. Wood, J. Henson, F. Love, W.R. Davis, P.D. Franzon, M. Bucher, S. Basavarajaiah, J. Oh, et al. Freepdk: An open-

source variation-aware design kit. In *Microelectronic Systems Education MSE'07, IEEE International Conference on*, pages 173–174, 2007.

[71] J.E. Stone, D. Gohara, and G. Shi. OpenCL: A parallel programming standard for heterogeneous computing systems. *Computing in science & engineering*, 12(3):66, 2010.

[72] H. Su, F. Liu, A. Devgan, E. Acar, and S. Nassif. Full chip leakage estimation considering power supply and temperature variations. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 78–83. ACM, 2003.

[73] K. Takeda and Y. Tohma. Logic design of fault-tolerant arithmetic units based on the data complementation strategy. In *10th Fault-Tolerant Computing Symposium*, pages 348–350, 1980.

[74] M. Tremblay, J. Chan, S. Chaudhry, AW Conigliam, and S.S. Tse. The majc architecture: A synthesis of parallelism and scalability. *Micro, IEEE*, 20(6):12–25, 2000.

[75] D.N. Truong, W.H. Cheng, T. Mohsenin, Z. Yu, A.T. Jacobson, G. Landge, M.J. Meeuwsen, C. Watnik, A.T. Tran, Z. Xiao, et al. A 167-processor computational platform in 65 nm cmos. *Solid-State Circuits, IEEE Journal of*, 44(4):1130–1144, 2009.

[76] J. Tschanz, K. Bowman, S. Walstra, M. Agostinelli, T. Karnik, and V. De. Tunable replica circuits and adaptive voltage-frequency techniques for

dynamic voltage, temperature, and aging variation tolerance. In *VLSI Circuits, 2009 Symposium on*, pages 112–113. IEEE, 2009.

[77] W.A. Wulf and C.G. Bell. C. mmp: a multi-mini-processor. In *Proceedings of the December 5-7, 1972, fall joint computer conference, part II*, pages 765–777. ACM, 1972.

[78] W. Zhao and Y. Cao. New generation of predictive technology model for sub-45 nm early design exploration. *Electron Devices, IEEE Transactions on*, 53(11):2816–2823, 2006.